VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Multimodální optimalizace pomocí evolučních algoritmů

Multimodal optimization via evolutionary algorithms

2013 Daniel Krpelík

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Zadání bakalářské práce

Student: **Daniel Krpelík**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 1103R031 Výpočetní matematika

Téma: Multimodální optimalizace pomocí evolučních algoritmů
Multimodal optimization via evolutionary algorithms

Zásady pro vypracování:

Cílem bakalářské práce je sestavit a odladit balík programů pro lokalizaci globálních extrémů funkcí mnoha reálných proměnných pomocí vybraných evolučních a pseudoevolučních algoritmů s reálnou reprezentací optimalizovaných proměnných. Provedeny budou i základní testy programů ve výpočtech stacionárních bodů na nadplochách potenciální energie modelových mnohočásticových soustav a porovnány s referenčními daty.

Postup práce:
1. seznámení se s problematikou a rešerše literatury,
2. návrh modulární struktury programového balíku,
3. sestavení optimalizačních programů dle zadané specifikace,
4. testovací výpočty.

Seznam doporučené odborné literatury:

[1] Zelinka I. a kol., Evoluční výpočetní techniky, BEN Technická literatura, Praha, 2009.
[2] Kvasnička V., Pospíchal J., Tiňo P., Evolučné algoritmy, Vydavateľstvo STU, Bratislava, 2000.
[3] R. L. Johnston Ed., Applications of Evolutionary Computation in Chemistry, Springer, Berlin, 2004.
[4] manuály k použitým implementacím programovacího jazyka FORTRANu
[5] internetové zdroje: http://www.wikipedia.org, http://www.-wales.ch.cam.ac.uk/CCD.html, http://apps.webofknowledge.com/, http://www.scopus.com/
[6] speciální časopisecká literatura

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. RNDr. René Kalus, Ph.D.**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

doc. RNDr. Jiří Bouchala, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2013

.....

Rád bych na tomto místě poděkoval doc. Renému Kalusovi za představení zajímavého problému a mé trpělivé vedení a směrování při práci na bakalářském projektu, doc. Jiřímu Dvorskému za poskytnutí šablony pro sazbu textu v prostředí LaTeXa všem ostatním, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Náplní práce je návrh a implementace a testování balíku globálně optimalizačních algoritmů hejnové inteligence pro hledání stabilních struktur atomových a molekulových klastrů, návrh testů pro porovnávání jejich kvality a programová implementace těchto testů a návrh několika základních vylepšení a porovnání jejich přínosu při řešení strukturální optimalizace atomových komplexů různých velikostí s využitím Lennardova-Jonesova interakčního modelu pro výpočet potenciální energie. Cílem je pak posouzení použitelnosti hejnových algoritmů pro optimalizaci struktur v molekulové fyzice a návrh dalšího postupu pro jejich vylepšení a uplatnění v praxi.

Klíčová slova: optimalizace, hejnová inteligence, klastr, Lennard-Jones

Abstract

The content of the work is to design and implement swarm intelligence global optimization algorithm package for the search for stable atomic and molecular cluster structures, design of their quality analysis and its implementation and several basic improvements are proposed and their benefits for solving structural optimization for atomic clusters of different sizes with potencial energy described by Lennard-Jones interaction model are assessed. The aim is to assess the applicatibility of swarm algorithms for structural optimization in molecular physics and to propose their further tuning and extensions for real application.

Keywords: optimization, swarm intelligence, cluster, Lennard-Jones

Seznam použitých zkratek a symbolů

ABC	–	Artificial Bee Colony (Umělá včelí kolonie)
CS	–	Cuckoo Search (Kukaččí algoritmus)
DE	–	Differencial Evolution (Diferenciální evoluce)
MCS	–	Modified Cuckoo Search (Upravený kukaččí algoritmus)
OpenMP	–	Open Multi-Processing
PES	–	Potential Energy Surface (Energetická nadplocha)
PSO	–	Particle Swarm Optimization (Optimalizace hejnem částic)
SOMA	–	Self-Organizing Migration Algorithm (Samo-organizační migrační algoritmus)

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Od chvíle, kdy se člověk poprvé dověděl o existenci atomů, molekul a geometrických vlastnostech jejich shluků, měl sen: Zjistit jaká pravidla určují tvary těchto struktur. Tato pravidla jsou dána interakčními silami. Existuje několik různých modelů mezi-atomových sil lišících se v přesnosti aproximace a reálném systému, který se snaží popsat. Je tedy možné, v různých přiblíženích, popsat potenciální energii různých struktur. Cílem ovšem nebylo jen vyčíslit energii známých struktur, ale prozkoumat vliv mezi-atomových sil na geometrická uspořádání molekul a jiných komplexů.

Zajímat se můžeme o několik základních typů atomových komplexů - volných atomů a sil působících mezi nimi, rigidních molekul, sestávajících z pevně vázaných atomů, které na sebe vzájemně působí silami slabšími, než jsou síly vázající vnitřní atomy a molekul kde mají i vnitřní atomy vlastní stupně volnosti, tedy slabší vnější síly mají omezený, ale nemalý vliv i na vnitřní stavbu jednotlivých molekul. Dále se budeme zabývat prvním typem, tedy shlukem atomů.

Stabilní molekulovou strukturu můžeme popsat jako takovou, která se nachází v lokálním minimu funkce poloh jednotlivých atomů popisující její vlastní energii $E_p(\boldsymbol{r_1}, ..., \boldsymbol{r_N})$ - nadplochy potenciální energie. Problémem nalezení lokálních minim se zabývá optimalizace.

Pro funkce definované na neprázdné množině prvků topologického prostoru můžeme definovat optimalizační úlohu, tedy nalezení jejich extrémů. Existují dvě základní optimalizační úlohy. Lokální optimalizace, která se zabývá hledáním lokálních extrémů a globální optimalizace, která se zabývá hledáním globálních extrémů na zadané množině.

Lokální optimalizace funkce na množině je tedy algoritmus, jehož výstupem je lokální extrém ležící v této množině. Lokální extrém je bod, pro který existuje takové jeho okolí, kde funkční hodnoty ve všech bodech tohoto okolí jsou menší nebo rovny funkční hodnotě v daném extrému. Existuje několik metod jak tento problém řešit. Pokud je funkce hladká, můžeme využít informace o jejím gradientu a navrhnout několik postupů, jak lokální extrém získat. Pro řešení je třeba najít stacionární body (body v níž je norma gradientu nulová) a tyto vyšetřit, jestli se jedná o lokální extrémy (z informací o vyšších derivacích případně analýzou okolí).

Může se stát, že neznáme analytický předpis funkce nebo je náročné najít a vyšetřit stacionární body. V takovémto případě je třeba použít numerické metody, které iteračně využívají známých informací o funkci a konvergují k hledanému řešení.

Globální optimalizace je složitější problém. Pro jeho analytické řešení je třeba znát všechny lokální extrémy na vnitřku množiny a extrémní hodnoty na hranici prohledávané množiny (pokud se jedná o uzavřenou množinu), tyto porovnat a jako výsledek prohlásit ten extremální.

Při použití numerických metod je tedy třeba nejprve numericky získat všechny lokální extrémy. Pokud by se nám podařilo získat konečnou množinu všech lokálních extrémů uvnitř množiny a všech lokálních extrémů na hranici prozkoumávané množiny, počítačově je porovnat už není problém.

Potíž s globální optimalizací je získat množinu všech lokálních extrémů. Pokud navíc máme jen omezené informace o optimalizované funkci nebo se naopak snažíme navrhnout postup použitelný u širší třídy různých funkcí, nemusíme vědět kolik extrémů máme

získat ani kde je máme hledat. V praxi se tedy nesnažíme nalézt množinu všech lokálních extrémů, ale přímo globální extrém.

V molekulové fyzice mají význam obě úlohy - lokální i globální optimalizace. Při lokální minimalizací vazební energie, hledáme tzv. metastabilní struktury. Pokud se struktura nachází v některém lokálním minimu, svou strukturu si zachová, dokud nedostane vnější energetický impuls. Množina všech takovýchto bodů i s jejich vazebními energiemi hraje významnou roli v termodynamice.

Dále můžeme lokálně minimalizovat normu gradientu funkce vazební energie. S využitím informací o vyšších derivacích nebo průzkumem okolí z těchto můžeme vyčlenit tzv. sedlové body. Tyto struktury a jejich vazební energie najdou využití při popisu dynamiky chemických reakcí, jelikož popisují energetické bariéry mezi dvěma stabilními stavy (tedy např. jakou energii je třeba dodat pro uskutečnění dané chemické reakce). Tyto struktury zahrnují jak stabilní konfigurace, tak sedlové body, které odpovídají metastabilním, přechodovým stavům.

Pokud se zajímáme o globální minimalizaci energetické nadplochy, hledáme strukturu s nejnižší potenciální energií. Pro popis termodynamiky shluku atomů nám často stačí znát několik různých konfigurací s nejnižšími energiemi, protože stavy s vyšší energií lze vzhledem k nízké pravděpodobnosti jejich výskytu zanedbat. Úkolem globální optimalizace v molekulové fyzice tedy není pouze nalézt konfiguraci s nejnižší energií, ale nalézt množinu konfigurací s energiemi kolem globálního minima. Cílem tedy není navrhnout algoritmus, který by při každém běhu nalezl přesné globální minimum, ale který by poskytoval různé struktury s nízkými energiemi.

Pro prohledávání prozkoumávané množiny se často využívá stochastických algoritmů, tedy prohledávání za pomocí náhody. Při uplatnění stochastických principů mohou dva běhy algoritmu se stejným nastavením poskytnou rozdílné výsledky. Využití náhody při optimalizaci lze realizovat i velmi jednoduše - pro náhodné počáteční body provedeme lokální optimalizaci některou spádovou metodou a najdeme jejich úpatí (lokální extrém). Vyvstává ovšem potřeba takovéto prohledávání zefektivnit. V minulém století začala vznikat spousta různých algoritmů, které se snaží vyhnout použití gradientní optimalizace, jelikož gradient musí být často počítán numericky a pro vysoko-dimenzionální problémy se stává nejnáročnější částí každé implementace. Přestože lokální optimalizace zůstává významnou součástí globální optimalizace, nastal rozvoj optimalizačních principů, které v některých aplikacích vykazují dobré výsledky i bez ní.

Základem pro tento rozvoj byly algoritmy „evoluční strategie“ [8] a „simulované žíhání“ [7,27]. Simulované žíhaní při prohledávání napodobuje princip relaxace při pomalém snižování teploty u reálného žíhání. Evoluční strategie využívá generování nových řešení abstraktně jako potomků předcházející generace. Právě z evolučních strategií vyplynula spousta metod s inspirací v živé přírodě. Zpočátku šlo o dynamiku evoluce v „genetických algoritmech“. Tento algoritmus se při prohledávání energetické nadplochy (PES) využíval hojně [17–26] a vzniklo proto několik vylepšení speciálně pro potřeby molekulové fyziky, které se využívají i v algoritmech navazujících. Návrat lokální optimalizace pak probíhá mj. u algoritmu „basin hopping“, který náhodně prohledává lokálně optimalizové struktury [29–32].

Pro inspiraci se sáhlo i do jiných odvětví, než evoluční biologie. Impulzem bylo, že samotné genetické algoritmy byly implementovány jako agentové algoritmy. Dalším krokem bylo podívat se na jiné přírodní agentové systémy, jmenovitě na různá živočišná

společenství. Vzniklo tedy odvětví hejnových algoritmů. Pro aplikaci na PES byly využity hlavně algoritmy na bázi optimalizace hejnem částic (*particle swarm optimization*, PSO) [10–13] a jeho úpravy [14–16], dále pak umělá včelí kolonie (*artificial bee colony*, ABC) [33–36,42] a diferenciální evoluce (*differential evolution*, DE) [37,38]. Každý z těchto algoritmů využívá upravenou verzi speciálně pro potřeby PES.

Základem pro porovnávání stability klastrových struktur je existence modelu popisujícího jejich potenciální energii. Modelů popisujících interakci mezi atomy a molekulami je mnoho (Lennard-Jones, Sutton-Chen, Gupta a další). V molekulové fyzice se liší převážně mírou aproximace. Dále také tím, které reálné molekuly dobře popisují, jelikož různé modely jsou různě úspěšné při popisu různých reálných systémů.

Energetická nadplocha (PES), což je funkce $\mathbb{R}^m \to \mathbb{R}$ daná interakčním modelem, udává potenciální energii molekulové struktury. Funkce závisí pouze na relativních polohách atomů[1] jako Lennard-Jonesův potenciál a je proto invariantní vůči translaci a rotaci a také k permutaci částic. Různé vektory tedy mohou reprezentovat ekvivalentní struktury.

Potíží s PES a jejím prohledáváním je několik. Počet lokálních extrémů roste exponenciálně s velikostí struktury, mění se jejich rozložení v prohledávaném prostoru a liší se výška bariér mezi jednotlivými extrémy [2]. Je tedy nutné použít algoritmy pro optimalizaci multimodálních funkcí. Není možné řešit problém hrubou silou, tedy prozkoumáním celého definičního oboru.

Optimalizační úloha je dána účelovou funkcí, která definuje prohledávaný nadprostor. Účelová funkce, někdy také označována za fitness funkci [6] nebo cenovou funkci [3, 6], reprezentuje kvalitu nalezených řešení. V základní verzi se při hledání stabilních klastrových struktur se může jednat o potenciální energii dané struktury.

Pro potřeby optimalizace lze tuto funkci upravovat. Při tomto je ovšem třeba zachovat uspořádání prvků, tedy pokud určitá struktura má nižší energii, než jiná, je žádoucí, aby tato vlastnost byla zachována i u upravené účelové funkce.

Důvodem pro úpravu už uspořádaného oboru hodnot energetické funkce může být například škálování - úprava metriky. Některé algoritmy využívají ruletovou selekci [6] a při změně účelové funkce můžeme značně ovlivnit pravděpodobnost výběru daného řešení. Při optimalizaci vektorových funkcí je třeba zavést zobrazení jejich oboru hodnot na uspořádanou množinu jako při hledání sedlových bodů, kdy normujeme gradient nebo při vícekriteriální optimalizaci, kdy je třeba popsat množinu optimálních výsledků (tzv. Paretovu množinu [6]).

Cílem této bakalářské práce je porovnat schopnosti vybraných hejnových optimalizačních algoritmů vyřešit problém strukturální optimalizace, navrhnout testy pro jejich porovnání a pro základní verze algoritmů je provést. Uvažujeme úlohu optimalizace pro klastry identických částic různých velikostí. Pro výpočet potenciální energie struktury je použit Lennard-Jonesův potenciál. Optimalizace probíhá s cílem nalézt strukturu s minimální potenciální energií.

V práci nejdříve uvedu základní pojmy a některé vztahy z oblasti optimalizace, kterých využívám pro popis řešené úlohy (kapitola 2.1), definuji problém, na němž algoritmy testuji, (kapitola 2.2), popisy a základní vlastnosti použitých hejnových algoritmů, a

[1] pokud neuvažujeme jinou úlohu, která by se např. zajímala o optimalizaci struktur ve vnějších silových polích

popis samotné programové implementace (kapitola 2.7), srovnání výsledků provedených numerických testů (kapitola 3) a shrnutí výsledků a návrhy pro navazující postup při vývoji a klasifikaci obdobných algoritmů (kapitola 4).

2 Teoretická část

2.1 Optimalizace

Hledání minim potenciální energie vede na optimalizační úlohu. I malé struktury ovšem vedou na vysoce-dimenzionální optimalizační problém. Tato dimenze roste lineárně s počtem částic struktury. Samotný výpočet energie může být také velmi náročnou operací. Optimalizace atomových a molekulových struktur je tedy úzce svázána s rozvojem výpočetní techniky a numerických metod ji realizujících.

Definice 2.1 *Mějme dánu funkci $f : D \to \mathbb{R}$, kde D je otevřená množina prvků separabilního vektorového prostoru se skalárním součinem. Optimalizační (minimalizační) úlohou nazveme problém hledání bodu $\boldsymbol{c} \in D$.*

Bod $\boldsymbol{c} \in D$ nazveme globálním řešením (optimem) funkce f na D, pokud platí:

$$\forall \boldsymbol{x} \in D : f(\boldsymbol{c}) \leq f(\boldsymbol{x})$$

Bod $\boldsymbol{c} \in D$ nazveme lokálním řešením (extrémem) funkce f na D, pokud existuje takové okolí $O(\boldsymbol{c}) \subseteq D$ bodu $\boldsymbol{c}$ kdy platí:

$$\exists O(\boldsymbol{c}) \subseteq D(f) : \forall \boldsymbol{x} \in O(\boldsymbol{c}) : f(\boldsymbol{c}) \leq f(\boldsymbol{x})$$

Poznámka 2.1 (Souvislost s extrémy)

Problém nalezení řešení optimalizační úlohy je ekvivalentní s problémem nalezení extrémů. Pro lokální řešení tedy platí nutná podmínka existence lokálního extrému, tedy: Pokud je bod $\boldsymbol{c} \in D(f)$ lokálním extrémem funkce f, parciální derivace podle jednotlivých nezávislých souřadnic buď neexistují nebo jsou nulové [5].

2.1.1 Lokální optimalizace multimodálních funkcí

Definice 2.2 *(Křivka)*

Křivkou k na množině prvků D separabilního vektorového prostoru se skalárním součinem nazveme spojitou funkci

$$k : I \to D$$

kde definiční obor I křivky k je uzavřený interval [4].

Definice 2.3 *(Spádová množina)*

Nechť existuje funkce $f : D \to \mathbb{R}$.

Označme spádovou množinou bodu $\boldsymbol{x}$ takovou množinu $M_x \subseteq D$ maximální vzhledem k inkluzi, kde pro všechna $\boldsymbol{y} \in M_x$ existuje křivka $k : \langle 0, 1\rangle \to \overline{D}$ splňující:

- $k(0) = \boldsymbol{y}$
- $k(1) = \boldsymbol{x}$
- $\forall t_1, t_2 \in \langle 0, 1) : t_1 > t_2 \Rightarrow f(k(t_1)) \leq f(k(t_2))$

Poznámka 2.2 (Jednoznačnost a existence spádové množiny)
Jako maximální vzhledem k inkluzi je určena jednoznačně.
Lze si domyslet, že můžeme zadefinovat křivku i jako konstantní funkci, tedy pro každý bod $\boldsymbol{x} \in D$ je zaručena existence spádové množiny $M_x \subseteq D$.
Pro prvky z hranice otevřené množiny D ovšem spádová množina existovat nemusí.

Definice 2.4 *(Ekvivalence prvků podle funkce)*
Zavedeme ekvivalenci $\equiv_f$: $D \times D$ vůči funkci f tak, že dva prvky $\boldsymbol{y}, \boldsymbol{x} \in D$ budeme pokládat za ekvivalentní, právě tehdy když se jejich spádové množiny rovnají.

$$\boldsymbol{x} \equiv_f \boldsymbol{y} \Leftrightarrow M_x \subseteq M_y \wedge M_x \supseteq M_y$$

Důkaz. (Ekvivalence prvků podle funkce)
Pro $\boldsymbol{x}, \boldsymbol{y} \in D$ platí $M_y \subseteq M_x \Leftrightarrow \boldsymbol{y} \in M_x$. Protože pravá strana zaručí existenci křivky popsané v definici 2.3 z bodu $\boldsymbol{y}$ do bodu $\boldsymbol{x}$ a pro každý bod, pro který existuje taková křivka do bodu $\boldsymbol{y}$, pak křivku do bodu $\boldsymbol{x}$ můžeme slepit z těchto dvou.

- Reflexivita
  Pro bod $\boldsymbol{x}$ můžeme vytvořit křivku popsanou konstantní funkcí $k(t) = \boldsymbol{x}$. Pak spádová množina bodu obsahuje i samotný bod a ten je tedy ekvivalentní sám se sebou.

- Symetrie
  Pro $\boldsymbol{x}, \boldsymbol{y} \in D$ můžeme psát

  $$\boldsymbol{x} \in M_y \wedge \boldsymbol{y} \in M_x \Rightarrow \boldsymbol{y} \in M_x \wedge \boldsymbol{x} \in M_y$$

  Tedy

  $$\boldsymbol{x} \equiv_f \boldsymbol{y} \Rightarrow \boldsymbol{y} \equiv_f \boldsymbol{x}$$

- Tranzitivita

  $$\forall \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in D$$
  $$\boldsymbol{x} \equiv_f \boldsymbol{y} \wedge \boldsymbol{y} \equiv_f \boldsymbol{z} \Rightarrow$$
  $$\boldsymbol{x} \in M_y \wedge \boldsymbol{y} \in M_z$$

  Tedy existují popsané křivky z $\boldsymbol{x}$ do $\boldsymbol{y}$ a z $\boldsymbol{y}$ do $\boldsymbol{z}$. Platí tedy, že $\boldsymbol{x} \in M_z$.
  Obdobně ukážeme i opačnou implikaci a tedy ekvivalenci prvků $\boldsymbol{x}$ a $\boldsymbol{z}$.

■

Definice 2.5 *(Úplná množina lokálních extrémů funkce na množině)*
Množinu $\mathbb{L}_D \subseteq D$ nazveme úplnou množinou lokálních extrémů funkce f na D, právě tehdy když pro každý prvek $\boldsymbol{x} \in \mathbb{L}_D$ platí, že je lokálním řešením optimalizační úlohy pro funkci f na D a zároveň pro všechny prvky množiny $D \backslash \mathbb{L}_D$ platí, že žádný není lokálním řešením optimalizační úlohy pro funkci f na D.

Definice 2.6 *(Třídy ekvivalence lokálních extrémů funkce)*
Systém tříd ekvivalence na $\mathbb{L}_D$ vůči relaci $\equiv_f$ (definice 2.4) označíme $\mathbb{L}_D^*$.

Definice 2.7 *(Multimodální funkce)*
Funkci $f : D \to \mathbb{R}$ nazveme multimodální, pokud kardinalita systému množin $\mathbb{L}_D^*$ je větší, než jedna.
Tedy existuje více, než jedna třída ekvivalence lokálních extrémů funkce f.

Definice 2.8 *(Domény přitažlivosti lokálních extrémů funkce na množině)*
Nechť existuje funkce $f : D \to \mathbb{R}$.
Označme množinu $D^* \subseteq D$ největší (maximální vzhledem k inkluzi) podmnožinu D splňující:

- $\forall \boldsymbol{l}_1, \boldsymbol{l}_2 \in \mathbb{L}_D : (M_{l_1} \cap M_{l_2}) \cap D^* \neq \emptyset \Leftrightarrow \boldsymbol{l}_1 \equiv_f \boldsymbol{l}_2$
- $\forall \boldsymbol{l} \in \mathbb{L}_D \forall \boldsymbol{y} \in \partial D : (M_l \cap M_y) \cap D^* \neq \emptyset \Leftrightarrow \boldsymbol{l} \equiv_f \boldsymbol{y}$
  kde ∂D je hranice množiny D

Množinu $B_l = (D^* \cap M_l)$ nazveme doménou přitažlivosti lokálního extrému $\boldsymbol{l}$ funkce f v D.

Poznámka 2.3 (O prázdných doménách přitažlivosti)
Nechť existuje funkce $f : D \to \mathbb{R}$.
Pak pro lokální extrém $\boldsymbol{l} \in \mathbb{L}_D$ doména přitažlivosti nemusí existovat.

Důkaz. (O prázdných doménách přitažlivosti)
Příklad :
Nechť $g : (a, e) \to \mathbb{R}$ je spojitá funkce dána svými vlastnostmi:
pro $a < b < c < d < e \in \mathbb{R}$, platí:

- g je klesající na (a, b)
- g je konstantní na $\langle b, c\rangle$
- g je klesající na (c, d)
- g je rostoucí na $\langle d, e)$

Pak každý bod z intervalu $\langle b, c)$ je lokálním extrémem a navíc bod d je rovněž lokálním extrémem. Podle definice 2.8 je spádová množina d rovna (a, e) a spádová množina každého lokálního extrému $l \in \langle b, c) : M_l = (a, c\rangle$ je její podmnožinou.
Největší množina $D^* \subseteq D$ tak, aby $(M_l \cap M_d) \cap D^* = \emptyset$, je tedy množina

$$D \backslash (a, c\rangle$$

a

$$B_l = M_l \cap D^* = \emptyset$$

protože

$$B_l = (a, c\rangle \cap ((a, e) \backslash (a, c\rangle) = \emptyset$$

■

Věta 2.1 *(Ekvivalence podle domén přitažlivosti)*
Nechť existuje funkce $f : D \to \mathbb{R}$.
Prvky $\boldsymbol{l}_1, \boldsymbol{l}_2 \in D$ jsou ekvivalentní podle relace $\equiv_f$ (definice 2.4), právě tehdy když se jejich neprázdné domény přitažlivosti rovnají.

Důkaz. (Ekvivalence podle domén přitažlivosti)

- $\Rightarrow$
  - $B_{l_1} \subseteq B_{l_2}$

    $$B_{l_1} \subseteq M_{l_1} \wedge B_{l_2} \subseteq M_{l_2}$$
    $$M_{l_1} = M_{l_2}, \forall(\emptyset \neq A) \subseteq D \Rightarrow M_{l_1} \cap M_{l_2} \cap A \neq \emptyset$$

    Tedy podle definice 2.8 jsou ekvivalentní.
  - $B_{l_1} \supseteq B_{l_2}$
    Obdobně.

- $\Leftarrow$
  Sporem:
  Nechť množina A je sjednocením domén přitažlivosti vůči funkci f všech $\boldsymbol{l} \in \mathbb{L}_D$.
  Předpoklad: mějme lokální extrémy $\boldsymbol{l}_1 \in L_1 \subseteq \mathbb{L}_D^*$ a $\boldsymbol{l}_2 \in L_2 \subseteq \mathbb{L}_D^*$ tak, že $L_1 \neq L_2$.
  Dále bod $\boldsymbol{x} \in A$ tak, že $\boldsymbol{x} \in B_{l_1} \wedge \boldsymbol{x} \in B_{l_2}$.
  Tedy $B_{l_1} \cap B_{l_2} \supseteq \boldsymbol{x} \neq \emptyset \wedge \neg(\boldsymbol{l}_1 \equiv_f \boldsymbol{l}_2)$.
  Podle definice domény přitažlivosti (2.8) a spádové množiny (2.3) platí:

  $$\forall \boldsymbol{l}_1, \boldsymbol{l}_2 \in \mathbb{L}_D : (M_{l_1} \cap M_{l_2}) \cap A \neq \emptyset \Leftrightarrow \boldsymbol{l}_1 \equiv_f \boldsymbol{l}_2$$

  ekvivalentně:

  $$\forall \boldsymbol{l}_1, \boldsymbol{l}_2 \in \mathbb{L}_D : (M_{l_1} \cap M_{l_2}) \cap A = \emptyset \Leftrightarrow \neg(\boldsymbol{l}_1 \equiv_f \boldsymbol{l}_2)$$

  tedy, protože $\boldsymbol{l}_1$ a $\boldsymbol{l}_2$ nejsou ekvivalentní podle $\equiv_f$, existuje prázdný průnik množin M_{l_1}, M_{l_2} a A. Jelikož podle definice 2.8 je doména přitažlivosti lokálního extrému $\boldsymbol{l} \in \mathbb{L}_D$ podmnožinou množiny M_l a zároveň $B_l \subseteq A$, tedy $B_l \subseteq M_l \cap A$ pak:

  $$(M_{l_1} \cap M_{l_2}) \cap A = \emptyset \Rightarrow (M_{l_1} \cap A) \cap (M_{l_2} \cap A) = \emptyset$$

  $$B_{l_1} \subseteq M_{l_1} \cap A \wedge B_{l_2} \subseteq M_{l_2} \cap A$$

  tedy

  $$B_{l_1} \cap B_{l_2} = \emptyset$$

  což je spor s předpokladem.

■

Věta 2.2 *(Domény přitažlivosti jako třídy ekvivalence)*
Nechť existuje funkce $f : D \to \mathbb{R}$.
Nechť množina A je sjednocením domén přitažlivosti vůči funkci f všech $\boldsymbol{l} \in \mathbb{L}_D$.
Relace $R : A \times A$ taková, že prvky $\boldsymbol{x}, \boldsymbol{y} \in A$ jsou v relaci, pokud leží v doméně přitažlivosti ekvivalentních lokálních extrémů (podle relace $\equiv_f$, definice 2.4), je relací ekvivalence na množině A.

Důkaz. (Domény přitažlivosti jako třídy ekvivalence)
Domény přitažlivosti jsou množiny, pro důkaz ekvivalence stačí dokázat, že prvky leží ve stejných množinách. Podle definice 2.8 žádný prvek $\boldsymbol{x} \in A$ neleží ve dvou různých doménách přitažlivosti.

- Reflexivita
  Každé tvrzení implikuje samo sebe, tedy pokud prvek leží v množině, tak leží v množině.
- Symetrie
  Logická konjunkce je komutativní, $\forall a, b \in B(\forall C \subseteq B : (a \in C \wedge b \in C) \Rightarrow (b \in C \wedge a \in C))$.
- Tranzitivita
  Prvky a a b leží v množině C a zároveň prvky b a c leží v množině B. Žádný prvek neleží ve dvou různých množinách. Jelikož $(b \in C \wedge b \in B)$, tak $C = B$, tedy $c \in C$, tedy a je v relaci s c.

■

Věta 2.3 *(O doméně přitažlivosti globálního extrému)*
Nechť existuje funkcionál $f : D \to \mathbb{R}$.
Pokud je $\boldsymbol{l} \in D$ globálním extrémem, jeho doména přitažlivosti je neprázdná.

Důkaz. (O doméně přitažlivosti globálního extrému)
Sporem: Předpokládejme, že bod $\boldsymbol{x} \in D$ je globálním extrémem na D, tedy

$$\forall \boldsymbol{y} \in D : f(\boldsymbol{x}) \leq f(\boldsymbol{y})$$

Dále, že neexistuje jeho doména přitažlivosti, tedy

$$\exists \boldsymbol{y} \in D : M_x \subseteq M_y \wedge M_x \neq M_y$$

Tedy existuje takový bod $\boldsymbol{y} \in \Omega(D)$ a křivka s nerostoucími funkčními hodnotami funkce f z bodu $\boldsymbol{x}$ do bodu $\boldsymbol{y}$, ale ne v opačném směru. To nutně znamená, že $f(\boldsymbol{y}) < f(\boldsymbol{x})$, což je spor s předpokladem. ■

Definice 2.9 *(Lokální optimalizace)*
Nechť existuje funkcionál $f : D \to \mathbb{R}$.
Nechť množina A je sjednocením domén přitažlivosti vůči funkci f všech lokálních extrémů $\boldsymbol{l} \in \mathbb{L}_D$.
Podle věty 2.2 existují třídy ekvivalence na A.
Lokální optimalizací s počátkem v bodě $\boldsymbol{x} \in A$ vzhledem k funkci f nazveme relaci $\mathbb{O} : A \times \mathbb{L}_D^*$ takovou, že $\forall \boldsymbol{l} \in \mathbb{L}_D^* : \boldsymbol{x} \in B_l \Rightarrow O(\boldsymbol{x}) = L$.
Řešením lokální optimalizace je množina lokálních minim ekvivalentních podle 2.4, kde pro všechna lokální minima z této množiny $\boldsymbol{x}$ leží v jejich doméně přitažlivosti.

Věta 2.4 *(Lokální optimalizace)*
Nechť existuje funkce $f : D \to \mathbb{R}$.
Nechť množina A je sjednocením domén přitažlivosti vůči funkci f všech lokálních extrémů $\boldsymbol{l} \in \mathbb{L}_D$.
Lokální optimalizace bodu $\boldsymbol{x} \in A$ funkce f je zobrazení.

Důkaz. (Lokální optimalizace je zobrazení)

- Existence : $\forall \boldsymbol{x} \in A(\exists L \in \mathbb{L}_D^* : (x, L) \in O)$
  Pokud je A prázdná, tvrzení platí triviálně. V opačném případě vyjdeme z definice domény přitažlivosti (definice 2.8), která zaručuje přítomnost lokálního extrému, v jehož doméně přitažlivosti x leží. Třídu ekvivalence získáme podle relace $\equiv_f$ (definice 2.4).

- Jednoznačnost : $\forall \boldsymbol{x} \in A(\forall L_1, L_2 \in \mathbb{L}_D^* : (((\boldsymbol{x}, L_1) \in O) \wedge ((\boldsymbol{x}, L_2) \in o) \Rightarrow L_1 \equiv_f L_2))$
  Sporem:
  Předpoklad : nechť existují dvě různé třídy ekvivalence lokálních extrémů L_1 a L_2, kteréžto jsou obě výsledkem lokální optimalizace bodu $\boldsymbol{x}$.
  Pak mají stejné domény přitažlivosti a podle věty 2.1 jsou prvky L_1 a L_2 ekvivalentní podle $\equiv_f$, což je spor.

■

Poznámka 2.4 (Body mimo domény přitažlivosti)

Numerické algoritmy lokální optimalizace jsou většinou implementovány jako spádové iterační algoritmy. V případě, že bod leží ve více spádových množinách, neplatí předchozí důkaz o jednoznačnosti. Různé algoritmy mohou podávat různé výsledky.V případě, že neexistuje lokální extrém, ke kterému by měl algoritmus konvergovat, nutně skončí na hranici definičního oboru optimalizované funkce.

Nechť existuje funkce $f : D \to \mathbb{R}$.

Nechť A je sjednocením domén přitažlivosti vůči funkci f všech lokálních extrémů $\boldsymbol{l} \in \mathbb{L}_D$.

Pak pro body množiny $D \backslash A$ můžou nastat dvě varianty:

Bod leží ve více spádových množinách různých lokálních extrémů. Nelze tedy jednoznačně určit výstup lokálně optimalizačního algoritmu.

V druhém případě neexistuje lokální extrém v jehož doméně přitažlivosti $\boldsymbol{x}$ leží. Optimalizace z takového bodu skončí na hranici prohledávané množiny. Tam se ovšem lokální extrémy nevyskytují (neexistuje okolí podle definice 2.1).

V praxi je ovšem nemožné ověřit, jestli je bod $\boldsymbol{x}$ validním vstupním bodem lokální optimalizace kvůli existenčním podmínkám v definicím. Lokální optimalizace se tedy provádí na celé množině D a její výsledky je třeba ověřovat. Případ, kdy prvek leží ve spádových množinách více lokálních extrémů a výstupem lokálně optimalizačního algoritmu bude jeden z nich při reálném nasazení skončí v lokálním extrému. Pokud výsledek neprojde testem pro nutnou podmínku lokálního extrému (viz poznámka 2.1), pak se jedná o prvek na hranici prohledávané množiny (nebo nedokonvergovaný výstup algoritmu).

2.1.2 Globální optimalizace multimodálních funkcí

Definice 2.10 *(Globální optimalizace)*

Nechť je dána funkce $f : D \to \mathbb{R}$.

Globální optimalizace je ekvivalentní s úlohou nalezení globálního extrému funkce f na množině D.

Definice 2.11 *(Redukovaná globální optimalizace)*
Nechť je dán funkcionál $f : D \to \mathbb{R}$.
Omezme se na globální extrémy, které jsou zároveň lokálními. Pak redukovanou globální optimalizací označme úlohu nalezení globálního extrému na největší otevřené podmnožině D.
Tedy nalézt $\boldsymbol{l} \in \mathbb{L}_D$ *takové, že* $\forall \boldsymbol{l}_1 \in \mathbb{L}_D : f(\boldsymbol{l}) \leq f(\boldsymbol{l}_1)$.

Poznámka 2.5 (Postup)
Se zavedenými pojmy ohledně klasifikace lokálních extrémů, jejich domén přitažlivosti a ekvivalence prvků v doméně přitažlivosti podle věty 2.2 můžeme při redukované globální optimalizaci postupovat takto:

- Nalézt libovolný prvek $\boldsymbol{x} \in D$ z domény přitažlivosti globálního extrému na $int(D)$ (podle věty 2.3 je tato množina neprázdná)

$$\boldsymbol{x} \in B_l$$

$$\forall \boldsymbol{l}_1 \in \mathbb{L}_D : f(\boldsymbol{l}) \leq f(\boldsymbol{l}_1)$$

- Lokálně optimalizovat nalezené x

$$\mathbb{L}_D^* \supseteq L = \mathbb{O}(x)$$

- Řešením je každý prvek $\boldsymbol{l} \in L$

Poznámka 2.6 (O spádových množinách)
Nestačí nalezení prvku ze spádové množiny globálního extrému, jelikož v takovém případě není výsledek lokální optimalizace jednoznačný a nemusí se tedy jednat o hledaný globální extrém.
V reálných aplikacích ovšem lokálně optimalizujeme na spádových množinách. Nalézt spádovou množinu globálního extrému je tedy mezikrok k řešení, jelikož doména přitažlivosti je její podmnožinou.

Poznámka 2.7 (Definiční obor)
Z důvodů programové implementace se jako definiční obor optimalizované funkce volí uzávěr omezené oblasti. Při volbě této oblasti se uplatňuje kombinace heuristického přístupu, kdy můžeme definiční obor omezit ze znalostí polohy hledaných řešení, a praktického přístupu, kdy je třeba množinu omezit, protože je výpočetně nemožné nebo neefektivní optimalizovat na původní množině.

Uvažujme dále funkcionál $f : int(D) \to \mathbb{R}$ a redukovanou globální optimalizaci.
Základními částmi globálně optimalizačních algoritmů jsou fáze prohledávání a dohledávání.
Cílem prohledávání je procházet prohledávanou množinu s úmyslem najít doménu přitažlivosti globálního extrému. Fáze dohledávání poté provede lokální optimalizaci a poskytne výsledek globální optimalizace. Globálně optimalizační algoritmy tedy musí zahrnout oba tyto principy pro nalezení výsledku.

Poznámka 2.8 Globální optimalizaci uváděnou dále v textu budeme považovat za definovanou redukovanou globální optimalizace.

2.2 Zkoumaný problém

Optimalizace s pravidly a pojmy, které jsme zavedli, umožní korektně aplikovat optimalizační algoritmy na problémy optimalizace struktur v molekulové fyzice. Při optimalizaci atomových struktur se zajímáme pouze o lokální extrémy. Pokud by se tedy výsledek optimalizace nacházel na hranici prohledávané množiny, není z fyzikálního hlediska relevantní, jelikož tyto hranice jsou nastaveny uměle.

Globální řešení optimalizační úlohy pro nějaký interakční model v molekulové fyzice udává dominantní strukturu při nulové termodynamické teplotě.

Cílem (redukované) globální optimalizace v molekulové fyzice je nalézt popsaný extrém. Reálné algoritmy globální optimalizace ovšem nemusí vždy nalézt přímo požadovaný extrém, ale jsou navrženy tak, aby byl rozdíl funkčních hodnot v nalezeném výsledku a požadovaném extrému minimální. Takovéto nesprávné výsledky jsou pak koproduktem reálného algoritmu globální optimalizace a také nalézají využití při popisu stavů, kdy je teplota systému nenulová.

2.2.1 Interakční model

Interakční model popisuje potenciální energii struktury.

Modelem uvažovaným v práci je Lennardův-Jonesův potenciál (vzorec 1) udávající interakční energii v závislosti na relativních polohách atomů.

$$E(x) = \sum_{\substack{\forall i,j \in [1,N] \\ i \neq j \\ i < j}} 4\epsilon \left(\left(\frac{\rho}{r_{i,j}} \right)^{12} - \left(\frac{\rho}{r_{i,j}} \right)^{6} \right) \tag{1}$$

Sčítání probíhá přes všechny různé dvojice z N elementů. Parametry ϵ (udávající hloubku minima) a ρ (udává vzdálenost dvou částic, při které je jejich vzájemná energie nulová) jsou nastavitelné a v mé práci nastaveny na hodnotu 1 z důvodu srovnání s [1].

2.2.2 Účelová funkce

Použitou účelovou funkcí je vlastní energie struktury dána interakčním modelem. Tedy

$$f(x) = E(x)$$

2.2.3 Prohledávaná množina

Pro optimalizační úlohu v molekulové fyzice můžeme prohledávanou množinu definovat jako otevřenou podmnožinu (kvůli vypuštění řešení na hranici) definičního oboru interakčního modelu (respektive účelové funkce, která interakční model využívá.) V kartézských souřadnicích je tato množina $V \subseteq \mathbb{R}^{3N}$, kde N je počet částic. Často volenými množinami bývají konvexní hyper-koule a hyper-kvádr. Omezení se zavádí pro zamezení vzniku disociovaných stavů, kdy při volbě příliš rozsáhlé množiny můžeme jako řešení optimalizace získat řešení pro menší systémy rozmístěné po této množině.

2.2.4 Stupně volnosti struktury

Potenciální energie E je ve zvoleném interakčním modelu funkcí relativních vzdáleností všech dvojic částic v systému. Je tedy invariantní vůči rotaci a posunutí celého systému. Pokud zvolíme trojrozměrný geometrický prostor, kde výsledky těchto dvou operací produkují odlišné prvky, vůči interakčnímu modelu můžeme zavést ekvivalenci prvků jako množinu všech, které jdou z jednoho na druhý těmito operacemi převést. Stupně volnosti udávají počet nezávislých proměnných, na kterých energie systému závisí. Můžeme jej chápat také jako minimální počet jednotlivých údajů, které potřebujeme pro jednoznačný popis struktury.

Představme si výstavbové pravidlo:

- 1. atom jednoznačně umístíme do počátku kartézské soustavy souřadnic
- 2. atom jednoznačně umístíme na kladnou poloosu x tak, že $||a_1 - a_2|| = r_{1,2}$
- 3. atom jednoznačně umístíme do I. kvadrantu roviny (x,y) tak, že $||a_1 - a_3|| = r_{1,3}$ a $||a_2 - a_3|| = r_{2,3}$
- umístění 4. a dalších atomů můžeme jednoznačně popsat trojrozměrným vektorem v kartézské bázi.

Potřebujeme tedy postupně (0,1,3,6,9,12 ...) souřadnic pro jednoznačný popis struktury.

Pro struktury velikosti $N > 3$ obecně platí : počet stupňů volnosti je $3 \cdot N - 6$, kde 6 je počet stupňů volnosti, vůči kterým jsou struktury invariantní.

2.2.5 Reprezentace struktury

Existuje více možných reprezentací molekulových struktur. Při volbě musíme počítat, že při použití $3 \cdot N$ souřadnic pro strukturu velikosti N vzniknou ve zvoleném prostoru třídy ekvivalence. Získáme tedy soustavu tzv. redundantních souřadnic.

Pokud navíc strukturu identických molekul reprezentujeme jako uspořádanou posloupnost jejich souřadnic v kartézském prostoru, obdržíme další ekvivalenci vůči permutaci této posloupnosti. Stejnou strukturu velikosti N (bez zohlednění předchozích geometrických invariancí) můžeme vyjádřit $N!$ různými způsoby.

Jednou z výhod těchto popisů je přehlednost a jednoduchost. Pokud bychom se chtěli vyhnout popsané permutační ekvivalenci, museli bychom například strukturu reprezentovat jako neuspořádanou množinu trojrozměrných vektorů s jejich násobnostmi, což by vyžadovalo složitější programovou implementaci.

Redundantní soustavy ovšem nejsou vždy na obtíž. Přestože geometrická pravděpodobnost náhodného nalezení některé třídy ekvivalence určité struktury zůstává stejná jako bez použití redundantních souřadnic, může tato redundantnost pomoci při prohledávání prostoru pomocí některých pravidel, jelikož každá struktura bude v tomto prostoru reprezentována jako $N!$ 6-ti rozměrných variet.

Pro struktury byla zvolena programová reprezentace vektorem polohových vektorů jednotlivých elementů v kartézském souřadnicovém systému. Elementy ve zkoumaných strukturách jsou atomy a poloha každého z nich je zapsána třemi prostorovými souřadnicemi. Struktura je tedy zapsána jako $3N$ hodnot reprezentujících polohy N různých atomů v kartézských souřadnicích.

Cílem globální optimalizace je nalézt takový lokální extrém l
O interakčních modelech víme následující:

- interakční potenciál je zdola spojitá funkce
- interakční potenciál je zdola omezená funkce
- limita potenciálu v nekonečnu je konečná hodnota
- existuje globální minimum, které je zároveň i minimem lokálním

Podle redukované Weirstassovy věty tedy na kompaktních podmnožinách definičního oboru daného interakčního modelu existuje globální minimum.

Podle posledního bodu také na vhodně zvolené otevřené podmnožině existuje globální minimum.

2.3 Hejnové algoritmy

Hejnové algoritmy jsou iterační stochastické multiagentové systémy. Nápad využít podobné algoritmy pochází z pozorování interakčních principů v reálných více-částicových systémů a živočišných společenství. Snaží se aplikovat přírodou optimalizované metody prohledávání pro zefektivnění prohledávání definičních oborů jiných optimalizovaných funkcí. Můžeme na ně nahlížet stejně jako na jejich předlohu tj. jako na dynamické systémy. Stavy tohoto systému jsou reprezentovány nalezenými řešeními, která uchovávají takzvaní agenti. Tito spolu komunikují a nová řešení jsou získávána jejich interakcí. Jeden iterační krok proto často bývá nazýván jako generace[2]. Jednotlivé hejnové algoritmy se krom své přírodní předlohy odlišují právě interakčními pravidly (někdy také nazývanými evoluční operátory).

Definice 2.12 *(Multi-agentový optimalizační systém)*
Multi-agentový systém je návrhový vzor využívající decentralizovaného zpracovávání informací s omezenými znalostmi. Sestává z:

- *množiny agentů (označme jako populaci)*
- *vnějšího prostředí*
- *systému pravidel*

Agenti řeší zadanou úlohu. Každý agent uchovává množinu vnitřních proměnných, které definují jeho chování. Při řešení optimalizace agenti mimo jiného udržují informace o nalezených řešeních.

Vnější prostředí definuje globální vlastnosti a proměnné, které jsou v různé míře přístupné agentům. U optimalizace se jedná např. o optimalizovanou funkci.

Systém pravidel definuje chování agentů. Určuje rozhraní pro komunikaci mezi jednotlivými agenty a vnějším prostředím a definuje pravidla pro úpravy stavů agentů. U optimalizace zde zahrnujeme např. evoluční operátory sloužící pro úpravu řešení agentů.

[2]Toto označení je zachováno z historických důvodů a vzešlo z algoritmů typu genetické evoluce a evoluční strategie, kdy agenti nové generace podle inspirace představovali potomky generace předchozí.

Definice 2.13 *(Uspořádání podle kvality)*
Pro agentový optimalizační algoritmus optimalizující funkci $f : D \to \mathbb{R}$ můžeme zavést relaci uspořádání indukované zobrazením f.

Agenta a nazveme kvalitnějším než agenta b, právě tehdy když pro řešení uchovávané agenty $r_a, r_b \in D$ platí:

$$f(r_a) < f(r_b)$$

Existuje několik základních principů, které bývají v různé míře implementovány ve většině algoritmů.

Definice 2.14 *(Elitismus)*
Elitismus je princip zachování kvalitních řešení.V evoluční biologii často popisován heslem : "Survival of the fittest."

Agenti pravidelně upravují svou polohu a může se stát, že se nalezené kvalitní řešení přepíše řešením horším. Některé algoritmy proto neupravují polohu nejlepšího řešení, jiné dokonce více takových.

Někdy může být ovšem výhodné takovéto řešení v globálně optimalizačních problémech opustit s úmyslem vymanit se z lokálního extrému. Tento další princip může být uplatněn například použitím Metropolisova kritéria [6].

Definice 2.15 *(Metropolisovo kritérium)*
Metropolisovo kritérium udává pravděpodobnost nahrazení kvalitnějšího řešení v populaci méně kvalitním podle vhodné funkce $p : H(f) \to \langle 0, 1 \rangle$.

Horší řešení je přijato s nenulovou pravděpodobností, kde hodnota může být založena např. na době, jakou algoritmus probíhá.

Definice 2.16 *(Vkládání náhodných řešení)*
Funkcionalita, kdy několik nejhorších řešení v každé iteraci je nahrazeno jinými řešeními (reálně je implementováno získávání nových řešení náhodně) pro zvýšení diverzity a zamezení předčasné homogenizaci (viz 2.3.1).

Hejnové algoritmy jsou ovšem velmi závislé na několika řídicích parametrech. Základním parametrem je velikost populace agentů. Další mají zpravidla různý vliv na evoluční operátory. Jejich nastavení je třeba provést před spuštěním algoritmu a v upravených variantách je možno některé měnit dynamicky. Můžeme vycházet z odhadů (ve svém programu využívám převážně tento způsob), nebo uvažovat o nastavení parametrů jako o další optimalizační úloze - pak se jedná o tzv. „meta-optimalizaci“. Pro meta-optimalizaci je tedy třeba vhodně zvolit optimalizační algoritmus a ještě důležitější část je dobře zvolit funkci, která bude reprezentovat kvalitu takto nastavených parametrů (více v kapitole 2.5).

2.3.1 Ukončovací podmínky

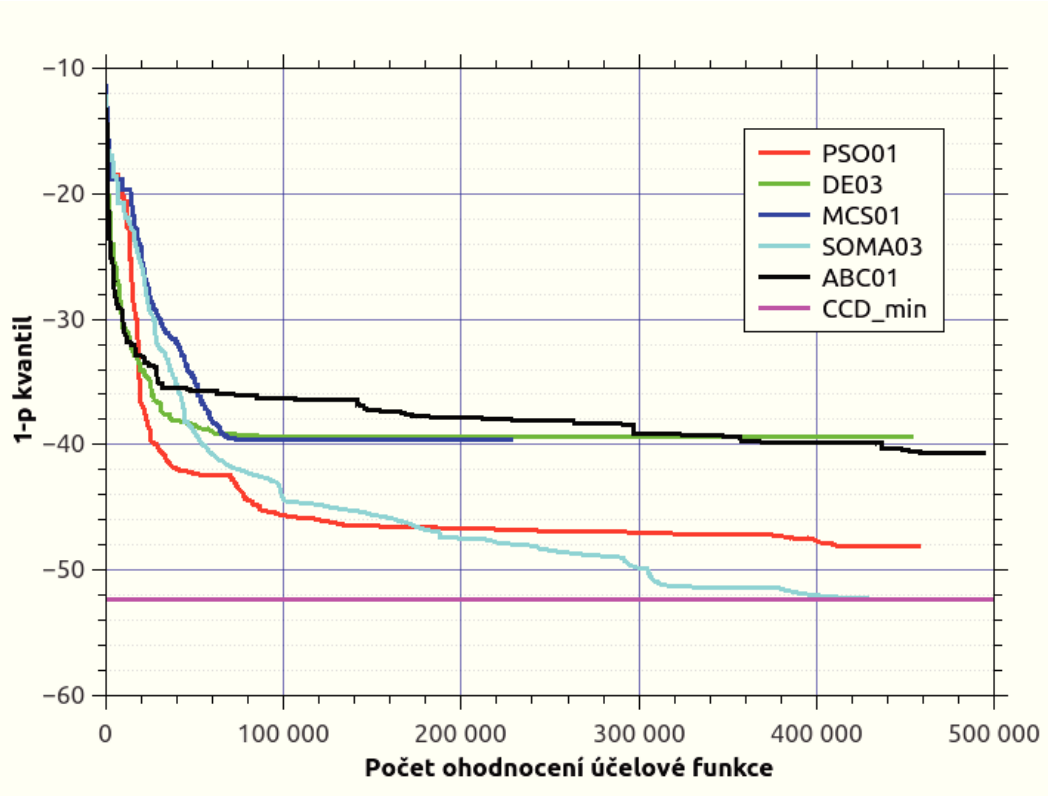
Jako u jiných optimalizačních metod můžeme ukončovat optimalizaci za pomoci informace o normě gradientu nebo provedení stanoveného počtu iterací. Agentový a iterační charakter algoritmů nám ovšem poskytuje další možnosti. Můžeme buď využít informací z předchozích iterací, kdy algoritmus ukončíme, pokud několik iterací nenalezl kvalitnější

řešení nebo informace o rozložení agentů. Populace se v průběhu běhu algoritmů shlukuje, proto impulzem pro ukončení může být tzv. „homogenizace populace“, kdy všichni agenti uchovávají stejné řešení.

Homogenizace znamená, že algoritmus ustrne a přestane dále prohledávat definiční obor optimalizované funkce. Tento stav můžeme identifikovat např. zavedením metriky na prohledávaném prostoru, určením průměru populace (největší vzdálenosti mezi řešeními dvou jedinců) a zavedením hraničního hodnoty, od které populaci považujeme za homogenizovanou.

Pokud označíme populaci jako homogenizovanou, můžeme buď ukončit algoritmus a započít další běh nebo populaci rozptýlit po okolí nalezeného řešení a pokračovat v prohledávání.

V programu jsem z důvodu použití specifických testů zvolil ukončení po určeném počtu iterací. Ukončovací podmínky jsem stanovil jako lineární funkci počtu částic. Jelikož se zabývám pouze malými systémy, tak je počet iterací dostatečný. Obrázek 1 ukazuje nalezené řešení algoritmem v závislosti na počtu ohodnocení účelové funkce pro klastr velikosti 15, náhodnou počáteční populací a posunem těžiště do počátku souřadnicového systému.



Obrázek 1: Průběh vývoje 1p kvantilu z 1000 nezávislých průběhů zvolených algoritmů v závislosti na počtu ohodnocení účelové funkce pro klastr velikosti 15 s náhodnou počáteční populací

2.3.2 Hranice prohledávaného prostoru

Množina, na které hledáme řešení, hraje významnou roli při funkčnosti algoritmu. U příliš rozsáhlé množiny spotřebováváme výpočetní čas na prohledávání oblastí s malou pravděpodobností výskytu globálního extrému. Při omezení hranice prohledávané množiny je třeba programově ošetřit jak postupovat, pokud by vzniklo řešení, které tento rozsah přesahuje.

Existují dva základní postupy, jak ošetřit řešení mimo definovanou množinu. Jedním je kontrola vzniklých řešení a při přesahu jejich navrácení zpět. To můžeme provést buď jejich posunutím přesně na hranici prostoru nebo uplatněním pružných srážek, kdy se struktura jakoby odrazí od hranice zpět podle vzorce

$$x_i = 2b_i - x_i$$

kde b_i je skalár reprezentující hranici i-té souřadnice a x_i je i-tá souřadnice vektoru reprezentujícího strukturu.

Druhou možností je penalizace na úrovni účelové funkce, kde se řešením, které prostor přesáhnou, uměle znehodnotí jejich účelová funkce. Tato penalizační funkce může být jak konstantní, tak záviset na vzdálenosti od hranice prohledávané množiny. Tento postup se může hodit, když předem neznáme velikost prohledávaného prostoru a chceme jej dynamicky měnit v průběhu běhu algoritmu.

V praxi se používají obě varianty. V práci byla použita metodu s pružným odrazem od hranice.

2.3.3 Rojení částic

Základním a jedním z prvních algoritmů je optimalizace rojením částic (particle swarm optimization, PSO) [10]. Inspiruje se pohybem ptačích a rybích hejn, kdy existuje vedoucí jedinec reprezentující zatím nejlepší nalezené řešení, kterého ostatní následují, přičemž využívají i svých vlastních znalostí (své zatím nejlepší nalezené řešení a historii pohybu po prohledávané množině). Původně byl využit pro simulaci pohybu virtuálních hejn (např. nově pro simulaci letu netopýrů ve filmu „Batman begins“). Ukázalo se, že takový pohyb může být výhodný při globálně optimalizačních problémech. Agenti krom nalezeného řešení uchovávají také informace o svém pohybu v podobě nejlepšího předchozího řešení (mezi těmito musí nutně být i nejlepší zatím nalezené řešení) a vektoru „rychlosti“ pohybu po prohledávaném prostoru (diferenciál polohy). V každé iteraci je jejich okamžitá rychlost upravena podle působících „sil“. Tyto se často označují za globální a lokální (bez souvislosti s použitím těchto termínů dříve při definici optimalizační úlohy). Globální síla působí směrem k nejlepšímu nalezenému řešení a lokální směrem k nejlepšímu dříve nalezenému řešení daným agentem. Pro zlepšení prohledávání je přidána síla působící směrem k nejlepšímu jedinci. Po úpravě rychlosti podle vzorce 2 se upraví řešení uchovávané agentem podle vzorce 3.

$$\boldsymbol{v}_{k+1} = w \cdot \boldsymbol{v}_k + c_1 \cdot r() \cdot (\boldsymbol{x}_{globalbest} - \boldsymbol{x}) + c_2 \cdot r() \cdot (\boldsymbol{x}_{localbest} - \boldsymbol{x}) + c_3 \cdot r() \cdot (\boldsymbol{x}_{bestinpop} - \boldsymbol{x}) \quad (2)$$

kde $\boldsymbol{v}$ značí uvedenou rychlost, $r()$ je generátor čísel z intervalu $(0, 1)$ s uniformním náhodným rozdělením a w, c_1, c_2 a c_3 jsou nastavitelné parametry algoritmu.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{v}_{k+1} \quad (3)$$

kde $\boldsymbol{x}_i$ udává řešení uchovávané agentem v i-té generaci.

Nastavení parametrů při testech je provedeno podle tabulky 1.

Pseudokód :

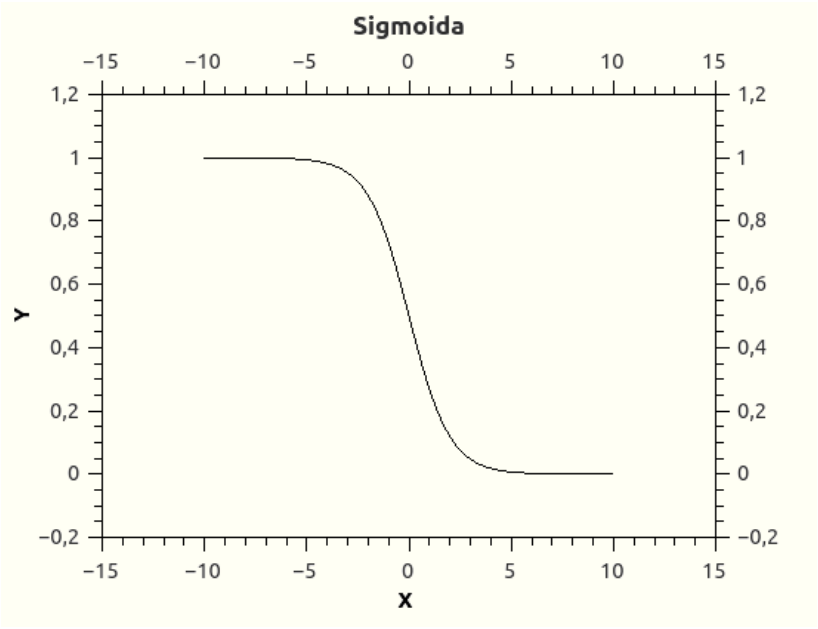
```
INIT:
   P = počáteční populace
   l = P
   bsf = nejlepší jedinec z P
   v = náhodné vektory

REPEAT:
do while(nejsou splněny ukončovací podmínky)
   b = nejlepší jedinec z P
   do i = 1, velikost populace
      h1 = w·v(i)
      h2 = c1·random(0,1)·(bsf−P(i))
      h3 = c2·random(0,1)·(l(i)−P(i))
      h4 = c3·random(0,1)·(b−P(i))
      v(i) = h1 + h2 + h3 + h4
      kandidát = P(i) + v(i)
      if (f(kandidát) < f(P(i))) then
         P(i) = kandidát
      endif
      if (f(P(i) < f(bsf)) then
         bsf = P(i)
      endif
      if (f(P(i) < f(l(i))) then
         l(i) = P(i)
      endif
      enddo
   enddo
enddo
```

Výpis 1: Pseudokód PSO

2.3.4 Umělá včelí kolonie

Dalším používaným algoritmem v optimalizacích struktur je umělá včelí kolonie (artificial bee colony, ABC) [33]. Jak z názvu vyplývá, inspirace vzešla ze včelích společenstev. V základním algoritmu existují tři druhy včel. Dělnice (*employment bees*), které udržují nalezená řešení, hledači (*onlooker bees*), kteří provádějí prohledávání v okolí nalezených řešení a průzkumníci (*scout bees*), kteří prohledávají prostor nezávisle na nalezených řešeních. Hledači si vybírají jednu z employment bees tzv. „ruletovou selekcí“ a v jejím okolí pak hledají nová řešení.

Ruletová selekce je princip náhodného výběru podle kvality, kdy s největší pravděpodobností je vybrán prvek s nejlepší kvalitou, ovšem s nenulovou pravděpodobností může být vybrán i prvek s nejhorší kvalitou. Pravděpodobnost výběru je v praxi často určována sigmoidální funkcí $y_a = \frac{1}{1+e^{k \cdot f(x_a)}}$, kde $f(x_a)$ je funkční hodnota řešení uchovávaného agentem a. Je zaručena vyšší pravděpodobnost výběru kvalitnějších řešení a je také možno jednoduchou lineární transformací upravit rozdíly mezi pravděpodobností výběru nejlepšího a nejhoršího prvku přeškálováním zobrazení funkčních hodnot řešení nalezených jednotlivými agenty na definiční obor sigmoidální funkce.



Obrázek 2: Průběh sigmoidální funkce pro k=1.

Pravděpodobnost výběru agenta a je pak dána vzorcem

$$p_a = \frac{y_a}{\sum_{i=1}^{N} y_i}$$

kde N je počet agentů v populaci.

Z implementačního hlediska je populace tvořena dělnicemi. Nastavený počet hledačů udává počet ruletových výběrů a následných úprav daného agenta. Tyto probíhají náhodným (tentokrát uniformním) výběrem jiného agenta z populace a vytvořením řešení podle vzorce

$$\boldsymbol{x}_{new} = \boldsymbol{x} + rand(-1, 2) \cdot (\boldsymbol{x}_2 - \boldsymbol{x})$$

kde $rand(0, 1)$ je generátor čísel z uniformního náhodného rozdělení.

Přijetí nového řešení je podmíněno jeho lepší kvalitou.

Každý agent udržuje vnitřní počitadlo, které udává kolikrát byl upraven bez přijetí nového řešení. Překročení určitého nastaveného limitu indikuje, že získání lepšího řešení z tohoto agenta je nepravděpodobné a je proto nahrazen řešením přijatým od průzkumníků. Prakticky je možno použít náhodný vektor, případně nechat malou populaci průzkumníků samostatně prohledávat prostor různými metodami od využití jiného algoritmu až po náhodné prohledávání.

Pro lepší globální prohledávání je také uplatněn princip, kdy je v každé generaci několik nejhorších řešení nahrazeno řešeními nalezenými prohledávači. Velikost části nahrazované populace je definována řídicím parametrem.

Nastavení parametrů při testech probíhá podle tabulky 2.

Pseudokód :

```
INIT:
   P = počáteční populace
   cycles = σ

REPEAT:
do while(nejsou splněny ukončovací podmínky)
   c = σ
   do i = 1,OBcount
      n = náhodný index prvku z P vybraný podle ruletové selekce
      c(n) = c(n) + 1
   enddo
   do i = 1, velikost populace
      do j = 1,c(i)
         x = náhodný prvek z P
         kandidát = P(i) + random(−1,2)·(x−P(i))
         if (f(kandidát) < f(P(i))) then
            P(i) = kandidát
            cycles(i) = 0
         else
            cycles(i) = cycles(i) + 1
         endif
      enddo
      if (cycles(i) > mC) then
         P(i) = řešení od průzkumníka
      endif
   enddo
   do (pro všechna x∈ pA část nejhorších řešení)
      x = řešení od průzkumníka
   enddo
enddo
```

Výpis 2: Pseudokód ABC

2.3.5 Samoorganizační migrační algoritmus

Samo-organizační migrační algoritmus (self-organizing migration algorithm, SOMA) se inspiruje chováním složitějších společenstev, jmenovitě lidí. Existuje více variant, ale v té základní opět existuje vedoucí jedinec, kterého všichni následují a putují k němu, ovšem ne vždy přímo. Každý jedinec vytvoří vektor podle vedoucího jedince a prozkoumá několik řešení v tomto směru. V základní verzi jde o pevný počet pokusů v pevně daných polohách podle vzorce 4. Stochastický prvek v tomto algoritmu je uplatněn právě volbou tohoto směru, kdy se podle nastaveného parametru náhodně určuje podprostor, ve kterém bude daný jedinec putovat. Každý bázový vektor původního prostoru je do daného podprostoru vybrán s nastavenou pravděpodobností (cR).

$$\boldsymbol{x}_k = \boldsymbol{x} + k\frac{c}{d}(P(\boldsymbol{x}_{best} - \boldsymbol{x})) \tag{4}$$

kde k je k-té zkoumané řešení a c a d jsou nastavitelné parametry : c určuje relativní délku vektoru k nejlepšímu řešení, d je počet zkoumaných řešení a $P()$ projekce na podprostor generovaný náhodně vybranými bázemi původního prostoru.

Nastavení parametrů při testech je provedeno podle tabulky 3.

Pseudokód :

```
INIT:
   P = počáteční populace

REPEAT:
do while(nejsou splněny ukončovací podmínky)
   b = nejlepší jedinec z P
   do i = 1, velikost populace
      do j=1,d
         do k=1,dimenze problému
            if (random(0,1) < cR) then
               kandidat(j)_k = P(i) + j·c/d(b−P(i))
            else
               kandidat(j)_k = P(i)
            endif
         enddo
      enddo
      h = nejlepší jedinec z kandidátů
      if (f(h) < f(P(i))) then
         P(i) = h
      endif
      enddo
   enddo
enddo
```

Výpis 3: Pseudokód SOMA

2.3.6 Upravený kukaččí algoritmus

Upravený kukaččí algoritmus (modified cuckoo search, MCS) [41] vznikl z kukaččího algoritmu [40]. Ten prováděl lokální prohledávání kolem jednotlivých řešení a tato ukládal do náhodně zvoleného agenta, pokud byla kvalitnější podle chování kukaček, která kladou vajíčka do hnízd jiných ptáků, které jejich potomci „ovládnou“, pokud jsou lepší než původní obyvatelé. Modifikace k těmto principům přidává interakci mezi agenty, kdy několik nejlepších jedinců je kombinováno mezi sebou. Tento algoritmus tedy zachovává fázi lokálního prohledávání a přidává operátory pro kombinaci řešení.

Pro zvolenou část nejlepších řešení (relativní část w) z populace je postup získávání nových řešení následující:

Pro každého jedince se náhodně zvolí jeden z těchto elitních jedinců. Pokud jsou různí provede se jejich kombinace podle vzorce

$$\boldsymbol{x} = \boldsymbol{x}_2 + \frac{\boldsymbol{x}_1 - \boldsymbol{x}_2}{\phi}$$

kde ϕ je hodnota $\frac{1+\sqrt{5}}{2}$, kde $\boldsymbol{x}_1$ je kvalitnější, než $\boldsymbol{x}_2$.

Pokud se jedná o stejné prvky, provede se lokální lokální prohledávání v okolí tohoto prvku s nastavením kroku A pro Lévyho let podle vzorce 5 ve vybraných směrech bázových vektorů (směr se vybere, s pravděpodobností cr). Pro zbytek populace probíhá lokální prohledávání s průměrem podle vzorce 6. Po vytvoření řešení se zvolí náhodný prvek z populace a pokud je nové řešení kvalitnější, staré se přepíše.

$$A = \frac{A_0}{k^2} \tag{5}$$

$$A = \frac{A_0}{\sqrt{k}} \tag{6}$$

A_0 je nastavitelný parametr a k je generace (iterace) algoritmu.

V implementaci byl Lévy let nahrazen náhodnou procházkou s logistickým rozdělením. Pro toto je nutno nastavit parametry pro střední hodnotu a rozptyl. Rozptyl zůstává nezměněn během iterací algoritmu a střední hodnota se upravuje, jak bylo navrženo pro krok Lévyho letu podle vzorců 5 a 6.

Jelikož velikost prohledávaného prostoru se s velikostí struktury mění, provádím nastavení střední hodnoty a rozptylu podle vzorců 7 a 8, kde mp a sp jsou řídící parametry algoritmu.

$$A_0 = \frac{b_{max} - b_{min}}{mp} \tag{7}$$

$$\sigma = \frac{b_{max} - b_{min}}{sp} \tag{8}$$

kde b označují hranice prohledávaného prostoru - pro můj problém jsou stejné pro každý prvek vektoru řešení.

Nastavení parametrů při probíhá testech podle tabulky 5.

Pseudokód :

```
INIT:
   P = počáteční populace
   A = A_0
   g = 0
REPEAT:
do while(nejsou splněny ukončovací podmínky)
   g = g + 1
   A = A_0 / g^2
   do (pro všechna x∈ w část nejlepších řešení)
      y = náhodně zvolený agent z w části nejlepších řešení
      if (x = y) then
         kandidát = Lévyho let z x s krokem délky (A)
         h = náhodný prvek z P
         if (f(kandidát) < f(h)) then
            h = kandidát
         endif
      else
         kandidát = x + (x+y)/φ
         h = náhodný prvek z P
         if (f(kandidát) < f(h)) then
            h = kandidát
         endif
      endif
   enddo
   A = A_0 / √g
   do (pro všechna x ze zbytku populace)
      kandidát = Lévyho let z x s krokem délky (A)
      h = náhodný prvek z P
      if (f(kandidát) < f(h)) then
         h = kandidát
      endif
   enddo
```

```
enddo
```

Výpis 4: Pseudokód MCS

2.3.7 Diferenciální evoluce

Na rozdíl od ostatních hejnových algoritmů není diferenciální evoluce (differential evolution, DE) [37] inspirována žádným přírodním společenstvím a většinou proto ani nebývá řazena mezi hejnové algoritmy. Jedná se o čistě abstraktní konstrukci pro výběr náhodných směrů pohybu. Pro každého každého agenta z populace a jeho uchovávané řešení $\boldsymbol{x_i}$ probíhá tvorba tzv. šumového vektoru (*noisy vector*) (ozn. $\boldsymbol{v}$) ze stanoveného počtu různých náhodně vybraných řešení z populace. Volitelné parametry upravují tvorbu nových řešení podle vzorce 9.

$$\boldsymbol{v} = \boldsymbol{x}_1 + F(\boldsymbol{x}_2 - \boldsymbol{x}_3) \tag{9}$$

kde F je nastavitelný parametr algoritmu.

Dále je šumový vektor $\boldsymbol{v}$ křížen s původním řešením podle vzorce 10 pro vznik nového řešení $\boldsymbol{u}$.

$$\boldsymbol{u}_k = \begin{cases} \boldsymbol{v}_k & rand(0,1) \leq Cr \\ \boldsymbol{x}_k & jinak \end{cases} \tag{10}$$

kde k je prvek vektoru, $rand()$ je náhodný generátor s uniformním rozdělením, $\boldsymbol{x}$ je původní řešení a Cr je nastavitelný parametr.

Zachovává se kvalitnější. Nastavení parametrů při testech provedeno podle tabulky 4.

Pseudokód :

```
INIT:
   P = počáteční populace

REPEAT:
do while(nejsou splněny ukončovací podmínky)
   do i = 1, velikost  populace
      r1 = náhodný prvek z P
      r2 = náhodný prvek z P
      r3 = náhodný prvek z P      \\ r1 ≠ r2 ≠ r3 ≠ r1
      do j = 1,dimenze problému
         if  (random(0,1) ≤ Cr) then
            uk = r1 + F·(r2 − r3)
         else
            uk = P(i)k
         endif
      enddo
      if  (f(u) < f(P(i)))  then
         P(i) = u
      endif
   enddo
enddo
```

Výpis 5: Pseudokód DE

2.4 Úpravy algoritmů

Věta „no free lunch theorem“ [6] tvrdí, že neexistuje algoritmus, který by byl vůči všem ostatním úspěšnější při řešení všech problémů. Věta postuluje ekvivalenci všech optimalizačních algoritmů, je-li jejich výkon posuzován průměrem přes všechny myslitelné optimalizační problémy. Z toho můžeme také vyvodit, že neexistuje algoritmus, který by dokázal efektivně řešit všechny optimalizační problémy. Podle [42] jsou dokonce základní verze hejnových algoritmů neefektivní pro optimalizaci struktury i malých klastrů. Proto se při uplatnění algoritmů na konkrétní problémy užívají specifická vylepšení. Tato jsou založena na heuristických znalostech o optimalizovaném problému a mají pomoci účelnějšímu prohledávání definičního oboru.

Ve své práci porovnávám několik základních vylepšení a jejich přínos pro kvalitu optimalizace.

2.4.1 Posun do těžiště

Cílem je zredukovat třídu ekvivalence způsobenou invarianci funkce potenciální energie vůči posunutí celé struktury. Těžiště struktury můžeme chápat jako těžiště $\boldsymbol{T}$ soustavy hmotných bodů.

$$\boldsymbol{T} = \frac{\sum_{i=1}^{N} \boldsymbol{x}}{N}$$

kde N je počet prvků struktury a $\boldsymbol{x}$ polohový vektor.

Pokud uplatníme transformaci, která těžiště posune do počátku soustavy kartézských souřadnic, tak tuto degeneraci omezíme.

2.4.2 Hlavní osy setrvačnosti

Po eliminaci invariance vůči translaci je třeba vyšetřit invarianci vůči rotaci. Cílem je tedy zredukovat zbylé třídy ekvivalence struktur (jak byly zavedeny v 2.4) do charakteristického reprezentanta každé třídy. Jelikož podmínka ekvivalence byla stanovena jako existenční, je v praktické implementaci těžko realizovatelná. Existují postupy k posuzování podobnosti struktur. Jednou možností je zavedení metriky (Kabschova metrika). Výpočet této metriky vyžaduje znalosti párových částic, tedy která částice z jedné struktury odpovídá částici v druhé struktuře. Metrika by se tedy musela určovat jako minimum přes všechna různá párování (bijekce N částic jedné struktury na částice druhé), kterých je $N!$.

Struktury, které lze na sebe převést rotací, mají stejné vlastní vektory tenzoru setrvačnosti. Moment setrvačnosti je fyzikální koeficient, který udává jaký odpor bude těleso konat při rotaci kolem zvolené osy. Pokud bychom chtěli strukturu rotovat kolem os kartézského souřadnicového systému, můžeme tento koeficient vypočíst. Pro každé tuhé těleso můžeme sestavit tzv. tenzor setrvačnosti, což bude matice 3x3. Tato matice bude vždy symetrická, pozitivně definitní. Výpočet tenzoru setrvačnosti probíhá podle vzorce 11.

Z teorie [9] víme, že pokud celou strukturu správně natočíme, nediagonální prvky vymizí. Toto nastane, pokud strukturu natočíme do tzv. hlavních os setrvačnosti. Výpočtem

těchto os, o kterých z teorie víme, že budou ortogonální, a daným natočením struktury můžeme opět zmírnit degenerace vzniklé kombinací řešení.

Hlavní osy setrvačnosti se hledají jako vlastní vektory zmíněného tenzoru setrvačnosti. Pokud vektor reprezentující strukturu v kartézském systému souřadnic přepíšeme do maticového tvaru, kde sloupce budou reprezentovat vektory poloh jednotlivých částic, natočení provedeme jednoduchým maticovým násobením podle vzorce

$$J(i,j) = \sum_{K=1}^{N} m_K \cdot \left(\left(\sum_{l=1}^{3} x_{K,l}^2 \right) \cdot \delta_{i,j} - x_{K,i} \cdot xK, j \right) \tag{11}$$

kde J je daný tenzor a $\delta_{i,j}$ je Kroneckerova delta funkce, sčítá se přes N částic, $x_{K,j}$ je j-tá souřadnice K-té částice a m_K je hmotnost K-té částice (částice atomových komplexů jsou totožné, používám tedy m_K = 1 pro všechny částice).

$$S_{nova} = E^T S_{stara} \tag{12}$$

kde S je popsaná matice reprezentující strukturu a E je matice, jejíž sloupce jsou nalezené vlastní vektory.

2.4.3 Indukční řešení

Při hledání optima soustavy dané velikosti může být vhodné využít předchozích znalostí získaných z řešení problémů pro menší velikosti. Pro jistý rozsah velikostí struktury může být zachováno výstavbové pravidlo, které může umožnit určit optimální řešení struktury velikosti N ze z optimálních struktur velikostí menších (zpravidla $N - 1$). Při generování náhodných řešení na počátku i v průběhu algoritmu můžeme tyto znalosti využít tak, že řešení pro menší systémy doplníme náhodně umístěnými částicemi. Takto vzniklou strukturu můžeme dále náhodně upravit (posunout, rotovat, permutovat) pro zvýšení diverzity. Článek [38] navíc navrhuje metodu chaotického generování počáteční populace, kdy první vektor je vytvořen z nedeformované známé struktury s přidáním náhodně umístěné částice a ostatní jsou generovány jeho deformací pro zvýšení diverzity populace.

Pro vkládání řešení z menších systémů nemusíme ovšem používat pouze globální minima. Při nárůstu velikosti systému může totiž docházet ke strukturálním přechodům, kdy výstavbové pravidlo z předchozích systémů přestává platit pro hledané globální optimum a naopak některé z předchozích lokálních optim může být strukturálně blízké právě hledanému globálnímu. Je proto vhodné zachovávat databázi o všech nalezených optimech (globálních i lokálních) a to, která z nich použít určovat na základě heuristických pravidel například stanovením pravděpodobnosti výběru jednotlivých prvků z nalezených řešení.

Tento princip ovšem není užitečný pouze při hledání optima pro určitý interakční model. Při řešení některého z výpočetně náročnějších interakčních modelů můžeme využít řešení nalezená modelem jednodušším. Které systémy jsou si podobné co do interakčních sil potom závisí na heuristických znalostech.

Pokud máme takovýto princip uplatněný a nemáme k dispozici žádná data o optimech, tak při řešení systému určité velikosti začneme nejprve řešením systémů menších, ukládáme výsledky a tyto použijeme v různé míře při generování náhodných řešení pro hledanou strukturu.

2.5 Meta-optimalizace

Hejnové algoritmy obsahují celou řadu nastavitelných parametrů a jejich účinnost může na konkrétním nastavení těchto parametrů významně záviset. Zmíněné parametry se mohou zadávat ručně, ale není těžké si rozmyslet, že samotné nastavení parametrů vede na další optimalizační úlohu. Otázkou ovšem je, jak zvolit účelovou funkci.

Tento přístup může být užitečný i při aplikaci na složitější interakční model. Pokud některý jednodušší model dobře popisuje chování složitějšího, můžeme meta-optimalizaci využít k nastavení parametrů pro tento složitější problém uplatněním meta-optimalizace na výpočetně méně náročný model

Důležitou vlastností stochastických algoritmů, kterou je mít na paměti, je jejich stochastická povaha. Proto při posuzování jejich kvality nemůžeme vycházet pouze z jednoho běhu, nýbrž sestavit statistiky, které by jejich kvalitu popisovaly. Navrhl jsem několik statistik, které je možno využít při optimalizaci nastavení řídících parametrů. Tyto jsou tvořeny po průběhu několika nezávislých běhů algoritmu z jimi nalezených optimálních řešení (z hodnot účelové funkce $f(x)$ v těchto nalezených řešeních). U všech je kladen důraz na to, aby se jednalo o skalární funkce možnost přímé aplikace optimalizačního algoritmu. Statistiky jsou inspirovány metodami pro porovnávání stochastických algoritmů.

2.5.1 Minimum

Základní statistikou při globální optimalizaci je nalezené optimum (zde minimum). Po provedení sady nezávislých testů se vybere nejnižší hodnota. Tato statistika má dvě úskalí. Jednak může jít o odlehlou hodnotu, tedy algoritmus se náhodou strefí do kvalitního bodu, i když ostatní běhy nacházejí mnohem horší řešení. Dále, pokud algoritmus funguje dobře pro více odlišných nastavení parametrů, mohly by tyto vést ke stejné minimální hodnotě, přestože některé by vykazovaly lepší charakteristiky (např. minimum najde více nezávislých běhů.)

Pro své účely používám mírně upravenou funkci, kde výsledná hodnota je udána vzorcem

$$s = \min(f(x)) + c \cdot CCD_{min}$$

kde $\min(f(x))$ je nejlepší nalezené řešení, CCD_{min} je tabulková hodnota (v tomto případě převzata z [1], jinak by se mohla určovat dynamicky v poměru nejlepšího řešení pro všechny běhy, případně s využití předchozích řešení) a c je relativní počet běhů, které se dostaly do předem stanovené vzdálenosti k tabulkové hodnotě.

Tato hodnota se neomezuje jen na nalezené minimum, ale zahrnuje také pravděpodobnost, s jakou bude toto minimum nalezeno.

2.5.2 Průměr

Další triviální statistikou je průměr všech běhů. Problémem je, že tato nám nic nevypovídá o minimu, které hledáme. Přesto udává průměrnou kvalitu nalezeného řešení a jako statistika více vypovídá o zvolených parametrech, než samotné minimum.

Cílem optimalizace je nalezení globálního extrému, proto je třeba zahrnout údaj o nejlepší nalezené hodnotě. Pomineme-li výše popsanou úpravu pro statistiku minima, podobné výsledky by mohla udávat kombinace minima a průměru podle vzorce

$$s = \alpha \cdot \min(f(x)) + (1 - \alpha) \cdot \text{mean}(f(x))$$

kde α je zvolený koeficient v rozmezí $< 0,1 >$ a mean$(f(x))$ je průměr nalezených výsledků.

Takováto kombinace statistik by opět vypovídá více o kvalitě zvolených parametrů.

2.5.3 Počet iterací

Pokud víme, že algoritmus je schopen minimum nalézt a my jej známe, můžeme se zajímat o to, aby jej nalezl co nejrychleji. V takovém případě nás bude zajímat statistika počtu kroků nutných pro nalezení minima. Ukončovací podmínkou je tedy

$$|f(x) - CCD_{min}| < \epsilon$$

kde CCD_{min} je známá hodnota účelové funkce optimálního řešení a ϵ je nastavitelná požadovaná vzdálenost od této hodnoty. Ze získaných počtů potřebných kroků pak tvoříme statistiky k porovnávání. V tomto případě se často užívá průměru potřebného počtu kroků.

2.5.4 Další možnosti

Toto zdaleka není kompletní výčet všech možných použitelných funkcí indikujících kvalitu nastavených parametrů. Jedná se jen o základ, který bývá používán při srovnávání algoritmů v literatuře. Z nalezených řešení můžeme tvořit různé statistiky a libovolně je kombinovat podle potřeby a požadovaného výstupu. Mimoto lze také využít jiných informací, které můžeme obdržet z nezávislých běhů algoritmů, pokud by se objevily nějaké směrodatné indikátory. Při optimalizaci vektorových funkcí (více, než jeden výstup, jako v případě kombinace minima a průměru v 2.5.2, kdy byla využita funkce pro převod na skalární hodnotu) je ovšem třeba definovat tzv. Paretovu množinu, což je množina řešení považovaných za optimální a zavést skalární funkci, po které dostaneme skalární optimalizační problém.

2.6 Lokální optimalizace

Jak bylo uvedeno na začátku kapitoly, je třeba do globálního algoritmu zahrnout jak fázi prohledávání, tak fázi dohledávání. Zmíněné algoritmy převážně zastávají fázi prohledávání a může se stát, že jejich výsledkem nebude přímo globální extrém. Je proto vhodné na konci každého běhu algoritmu výsledné řešení lokálně optimalizovat za účelem získání lokálního extrému.

Lokální optimalizaci můžeme využít také jako jeden z operátorů stochastických algoritmů. Potom provádíme globální optimalizaci na lokálně optimalizovaných strukturách [11, 38]. Při této aplikaci je nutno upravit řídící parametry pro snížení počtu prováděných lokálních optimalizací z důvodu její výpočetní náročnosti.

2.7 Implementace

2.7.1 Lokálně optimalizační algoritmus

Ve svém programu jsem pro lokální optimalizaci implementoval gradientní metodu sdružených gradientů [3] s mírnou úpravou algoritmu pro nalezení délky kroku ve

zvoleném směru. Tento je realizován stochastickým algoritmem SOMA (viz 2.3.5). Stochastický algoritmus jsem použil kvůli vlastnostem optimalizované funkce, která je silně nekvadratická a multimodální. Pro metody typu půlení intervalu bylo obtížné odhadnout počáteční délku intervalu tak, aby obsahoval pouze jeden hledaný extrém. Při aproximaci kvadratickou funkcí a pokusem iteračně hledat minima na základě aproximací Taylorovým polynomem byla potíž s nekvalitními aproximacemi. SOMA patří mezi algoritmy stochastické optimalizace, ale v jedné dimenzi se velmi blíží metodě půlení intervalu, ovšem díky svým vlastnostem dokáže opustit lokální extrémy.

Ukončovací podmínka je dána jednak požadovanou přesností, ale z důvodu konečného času pro testování také maximálním počtem iterací.

2.7.2 Architektura programu

Program je od počátku koncipován jako modulární pro pozdější využití na širší třídě úloh a testování více algoritmů a použitých úprav. Tato modularita ovšem spočívá na úrovni kompileru fortranovského kódu. Každá část programu je realizována v samostatném souboru a z programového hlediska v samostatném modulu. Mezi moduly existuje návaznost, kterou je třeba při kompilaci programu dodržet. Každý modul má na starost část programu a jejich záměnou je možno měnit řešený problém, použité algoritmy (jak optimalizační, tak vedlejší, starající se o vnitřní běh programu) nebo výstupy programu.

Z důvodu výpočetní náročnosti a množství prováděných testů je program paralelizován s použitím OpenMP a upraven pro možnost spouštění na výpočetních klastrech SPC VŠB-TUO.

Pro účely mé práce bylo navrženo několik variant modulů starajícího se o datové výstupy za účelem testování různých algoritmů, jejich úprav a jiných aspektů programu.

Účelem modulu **Interact** je počítat interakční energii. Jeho variabilita spočívá v jeho nahrazení jiným za účelem použití jiného modelu meziatomových interakcí. V práci je použit pouze Lennardův-Jonesův model daný vzorcem 1.

Modul **Optfnc** se stará o základní nastavení prohledávané množiny a účelové funkce. Určuje velikost vektoru řešení a tvoří interface mezi modulem interact a zbytkem programu. Vypočítává účelovou funkci a může ji různě modifikovat v závislosti na požadavcích na řešení (hledání minim a případně dalších stacionárních bodů jako jsou sedlové body 1. a vyšších řádů). Vždy je ale účelovou funkci třeba vyjádřit jako funkci skalární určenou pro hledání minima. Dále modul obsahuje rutiny pro výpočet derivace účelové funkce v daném bodě, generování náhodných vektorů a pro úpravu nalezených řešení (posun do těžiště, natáčení do os, průběžnou lokální optimalizaci apod.).

Úkolem modulu **Optglb** je poskytovat programu globální proměnné (parametry různých numerických algoritmů, dimenze optimalizované funkce atd.), známá řešení z [1] (nebo z předchozích běhů algoritmu - v budoucnu), případně zajištění interface s databází nalezených metastabilních struktur pro tvorbu počáteční populace.

Modul **Randgen** slouží ke generování náhodných čísel. Jeho rutiny jsou zařazeny do samostatného modulu z důvodu využití ve více částech programu a případné možnosti snadného nahrazení konkrétního typu generátoru.

Do pomocného modulu **Optlib** ukládám rutiny, které přímo nesouvisejí s globální optimalizací, ovšem jsou často využívány v průběhu algoritmu. Jsou to rutiny typu třídění pole, výpočtu derivací, generování speciálních datových struktur, použité numerické metody atd.

Modul **Optlocal** obsahuje jedinou funkci ve svém rozhraní a to pro lokální optimalizaci pro daný počáteční bod a maximální počet iterací. Různé parametry jako požadovaná přesnost normy gradientu jsou uloženy v „optglobal“. V současnosti je použita metoda sdružených gradientů (více v 2.1.1).

Modul **Optim** obsahuje globálně optimalizační algoritmus. Kvůli povaze testů jsou vnitřní proměnné globální a v modulu existují funkce pro inicializaci algoritmu a dále pro provedení jedné iterace. Toto je z důvodu, že některé testy vyžadují sběr dat v průběhu optimalizace.

Veškeré proměnné je nutno definovat jako „threadprivate“ kvůli rozhraní OpenMP použitém při paralelizaci běhů při testování.

Modul **Main** obsahuje samotný program, tedy definuje logiku optimalizace a datové výstupy. V tomto modulu jsou napsány veškeré prováděné testy. V tomto modulu je implementována paralelizace pomocí OpenMP pro spouštění více běhů algoritmů najednou.

3 Výpočetní část

3.1 Datové výstupy

Jelikož se jedná o stochastické algoritmy, nejsou výsledky z jednoho běhu algoritmu dostatečně prokazatelné. Většina sesbíraných dat jsou proto statistikami z více nezávislých běhů. Sběr dat probíhá způsobem, kdy každý algoritmus poskytne data o svém průběhu a tato pak podléhají různým dále popsaným analýzám. Základním výsledkem těchto testů ovšem není statistika, nýbrž nejlepší nalezená hodnota ze všech průběhů.

Jedná se o algoritmy iterační, proto se jako první nabízí analyzovat nalezené výsledky vzhledem k počtu iterací. Takovýto přístup by ovšem nebyl kompatibilní mezi jednotlivými algoritmy, kdy jedna iterace může být různě výpočetně náročná. Referenční množinou je proto zvolen počet ohodnocení účelové funkce. Hlavní důvod je také ten, že předpokládáme, že při nasazení algoritmů na složitější úlohy (např. při výpočtu potenciální energie pomocí kvantové chemie), bude tato část výpočtu nejnáročnější a ostatní (režijní) části algoritmu můžeme při odhadu náročnosti zanedbat.

Z dat poskytovanými jednotlivými nezávislými algoritmy jsou tvořeny základní statistiky : průměr, minimum, maximum, medián, dolní kvantil, horní kvantil a 1-p a 5-p procentní kvantily. Odlehlé hodnoty se nezanedbávají. Tyto statistiky mohou být tvořeny nad různými daty. V programu používám nejlepší nalezené řešení a dále pak parametr, který jistým způsobem vypovídá o diversitě populace agentů uvnitř algoritmu. Tato je popsána jako průměr směrodatných odchylek jednotlivých souřadnic z vektorů řešení napříč všemi agenty místo jiných z důvodu výpočetní náročnosti. Výstupem jsou pak mimo jiné grafy vykreslující závislosti těchto statistik.

Po ukončení iteračního cyklu jsou porovnány nalezené výsledky v několika nezávislých testech. Nalezené výsledky z nezávislých běhů jsou následně lokálně optimalizovány a pro data před a po lokální optimalizaci jsou provedeny tyto testy:

3.1.1 Nějlepší nalezený výsledek

Nalezený výsledek je porovnán se známými hodnotami z [1]. Jednak je zapsána jeho absolutní hodnota a jednak relativní hodnota vzhledem k výsledku z [1].

3.1.2 Statistiky výsledků

Výsledky (před a po lokální optimalizaci zvlášť) jsou porovnány s hodnotami z [1] s účelem zjistit, jaká část nezávislých běhů se dostane do jejich blízkosti. Jsou zapisovány relativní četnosti běhů, jejichž relativní odchylky od referenčních hodnot jsou menší, než 0.15, 0.1, 0.05, 0.025 a 0.01 relativní vzdálenosti k referenčním hodnotám z [1].

3.1.3 Závislost výsledku lokální optimalizace na výsledku globální optimalizace

Je vykreslen bodový graf korelace. Tento výsledek pomáhá graficky určit význam lokální optimalizace v závislosti na použitém algoritmu.

3.2 Nastavení řídicích parametrů algoritmů

V následujících tabulkách jsou uvedena nastavení řídicích parametrů použitých při numerických testech. Jednotlivá nastavení jsou označena akronymem pro příslušný algoritmus a svým pořadovým číslem (např. PSO01). Doporučené rozsahy jsou dány povahou parametru (např. rozsah $\langle 0, 1\rangle$ u parametrů udávající relativní části). Pro velikosti populace není udávána maximální hodnota, jelikož literatura zabývající se optimalizacemi v molekulové fyzice využívá průběžné lokální optimalizace a udávané hodnoty nejsou srovnatelné s metodami tuto úpravu nevyužívajícími. Literatura zabývající se danými algoritmy bez aplikace na konkrétní úlohy navrhuje tyto parametry nastavovat přímo úměrné dimenzi problému.

parametr	doporučený rozsah	PSO01	PSO02	PSO03	PSO04
velikost populace	> 1	150	50	150	97
w	$\langle 0, 1\rangle$	0.82	0.6	0.7	0.115
c_1	$\langle 0, 2\rangle$	1.5	2	0.5	2
c_2	$\langle 0, 2\rangle$	1.3055	2	0.5	0.8
c_3	$\langle 0, 2\rangle$	0.6	0	0.5	1.96

Tabulka 1: Nastavení parametrů PSO

parametr	doporučený rozsah	ABC01	ABC02	ABC03	ABC04
velikost populace	> 1	20	30	50	261
OBcount	> 1	100	300	1000	1308
pA	$\langle 0, 1\rangle$	0.4	0	0.4	0.41
mC	> 1	4200	50	800	6283
x_{min}	$\langle -10, 10\rangle$	-1	-0.8	-1	-0.56
x_{max}	$\langle -10, 10\rangle$	1	0.8	1	0.21

Tabulka 2: Nastavení parametrů ABC

parametr	doporučený rozsah	SOMA01	SOMA02	SOMA03
velikost populace	> 1	215	50	70
c	$(1, 5+\rangle$	1.053	1,478123	1.9
cR	$(0, 1)$	0.57	0.26	0.4
d	> 1	4	10	8

Tabulka 3: Nastavení parametrů SOMA

parametr	doporučený rozsah	DE01	DE02	DE03
velikost populace	> 1	400	50	60
cR	$(0, 1)$	0.5	0.2	0.8
F	$(0, 2)$	0.6	0.7	0.3

Tabulka 4: Nastavení parametrů DE

parametr	doporučený rozsah	MCS01	MCS02	MCS03
velikost populace	> 1	150	50	50
w	$\langle 0, 1\rangle$	0.3	0.6	0.7
cr	$(0, 1\rangle$	0.6	0.4	0.5
mp	-	100	1000	15000
sp	-	500	2000	10000

Tabulka 5: Nastavení parametrů MCS

3.3 Testování správnosti implementace

V tomto testu jsou vektory vstupní populace generovány ze známých řešení z [1]. Tyto prochází drobnými úpravami za účelem zjistit, jestli algoritmus z těchto mírně vychýlených řešení řešení konverguje zpět do známého optima. Deformace řešení probíhá náhodným posunutím prvků, náhodnou rotací struktury a následně náhodnou permutací částic. Test byl proveden pro všechny implementované algoritmy.

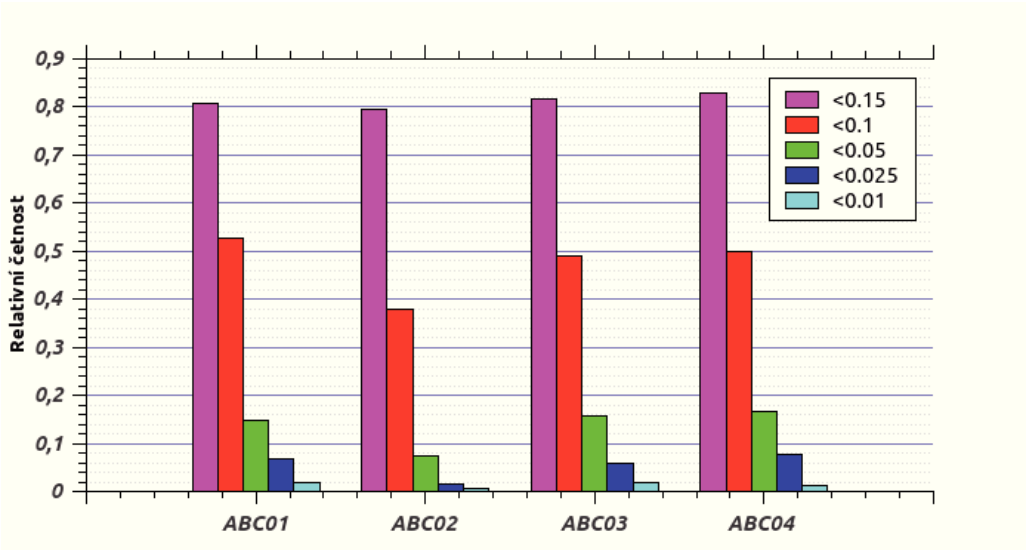
V tomto případě nejsou prezentovány žádné výsledky. Při testování se jedná jen o ověření správnosti implementace.

Mírně vychýlené optimální struktury v počáteční populaci skutečně vedou ke konvergenci zpět ke stabilní struktuře. V průběhu optimalizace navíc nedochází k degeneraci řešení, tedy funkce popisující kvalitu nalezeného řešení je nerostoucí (u minimalizační úlohy), tedy je správně implementován princip elitismu.

Algoritmy jsou tedy implementovány správně a můžeme dále posuzovat jejich výkon při uplatnění různých vylepšení.

3.4 Meta-optimalizované parametry

Pro algoritmus ABC byly analyzovány možnosti meta-optimalizace pro klastr velikosti 15, náhodnou počáteční populaci, velikost hrany hyperkrychle ohraničující prohledávanou množinu podle vzorce 15 a s posunem do těžiště bez finální lokální optimalizace (z důvodu snížení výpočetní náročnosti). Meta-optimalizace byla realizována algoritmem SOMA. Výsledky jsou porovnány s ostatními parametry pro dané algoritmy na obrázku 3.



Obrázek 3: Srovnání výsledku testu 3.1.2 pro algoritmus umělé včelí kolonie (ABC) a jeho různá nastavení. Algoritmy ABC01-ABC03 jsou nastaveny ručně, ABC04 vznikl po aplikaci meta-optimalizace.

Z obrázku 3 je vidět, že při použití meta-optimalizovaných parametrů nedosáhneme významně lepších ani horších výsledků vůči heuristickému nastavení. Toto může být způsobeno jednak absencí finální lokální optimalizace při realizaci meta-optimalizace, tedy meta-optimalizací jiné úlohy. Z obrázku 9 totiž vyplývá, že pro algoritmus ABC neexistuje korelace mezi kvalitou řešení před fází dohledávání a po ní.

Existuje tedy vícero optimálních nastavení parametrů. Přesto může být zásadní tento test provést při testování upravených algoritmů, kdy může být složité řídící parametry heuristicky nastavit.

3.5 Vliv počáteční populace

Pro algoritmus ABC analyzuji vliv počáteční populace na celkový výsledek optimalizace. Testy jsou provedeny pro zvolené velikosti klastrů. Počáteční populace jsou tvořeny různými pravidly, kde je snaha o využití heuristických poznatků o řešené úloze.

Řešení jsou generována vkládáním náhodných částic do deformovaných optimálních struktur menší velikosti; vkládáním náhodné částice do deformované optimální struktury předešlé velikosti. Cílem je porovnat, jestli účelný výběr počáteční populace může zlepšit výkon algoritmu.

Obrázky 4, 5 a 6 srovnávají výsledky pro různě velké klastry a různé počáteční populace, kde deformovanou populací je myšlena populace tvořena deformovanými optimálními strukturami klastru stejné velikosti (převzaty z [1]). Stejná míra deformace je totiž použita i při vkládání známých menších struktur.

Deformace struktury je prováděna podle vzorce 13.

$$\boldsymbol{x}_{new} = perm(rot(\boldsymbol{x}_{old} + rand(0.2, 1.2) \cdot k \cdot \boldsymbol{r})) \tag{13}$$

kde $\boldsymbol{x}$ reprezentuje vektor struktury, $rand()$ je náhodný generátor s uniformním rozdělením, k je nastavitelný parametr a $\boldsymbol{r}$ je náhodný jednotkový vektor. Funkce $perm()$ a $rot()$ pak

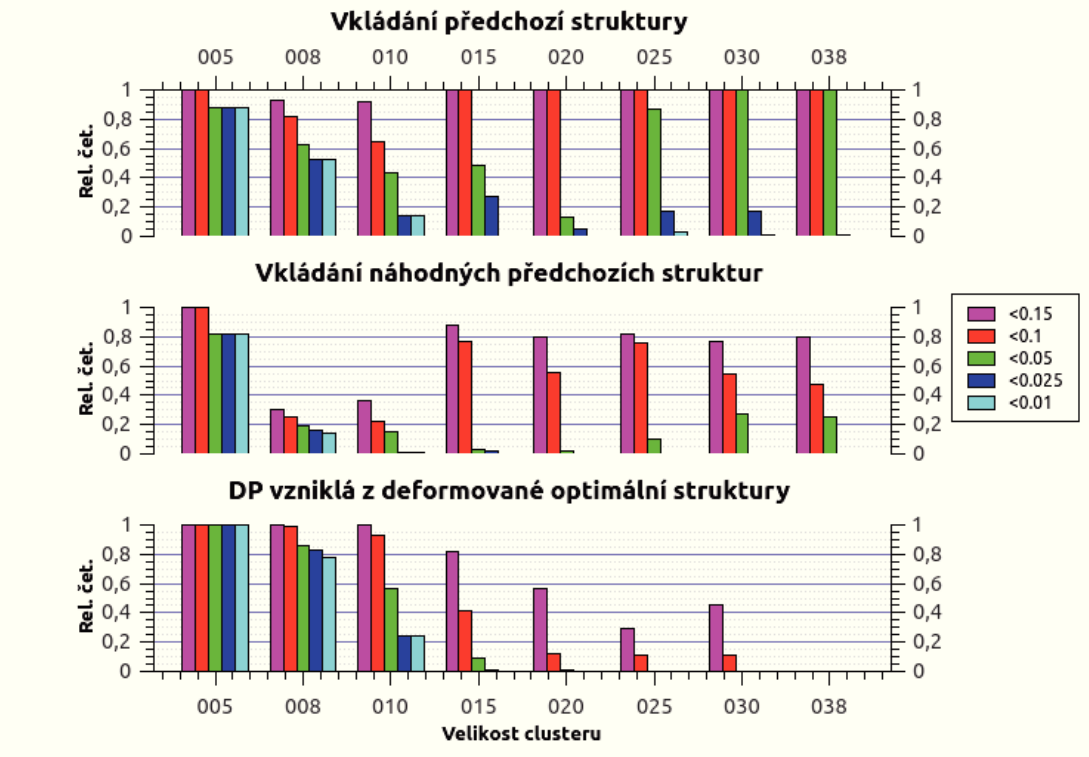
představují funkce pro náhodnou permutaci částic a rotaci celé struktury. Parametr k nastavuji v závislosti na velikosti struktury podle vzorce 14, který vznikl řízenými pokusy pro hledání takového vztahu, který by reprezentoval dostatečnou deformaci pro absenci podobnosti mezi deformovanou a původní strukturou.

$$k = \log(N) \cdot \sqrt{\frac{N}{10} + \log(N)} \tag{14}$$

kde N je počet částic systému.

Pro test na obrázku 4 jsem použil deformaci velikosti k, pro test na obrázku 5 deformaci velikosti $\frac{k}{2}$ a pro test na obrázku 6 žádnou deformaci.

Poslední okno grafů 4 a 5 tedy vyjadřuje schopnost algoritmu dokonvergovat zpět do optima po deformaci známé optimální struktury stejnou měrou, jaké deformace bylo použito při deformaci známých struktur vkládaných do počátečních řešení. Z výsledků jde vidět, že i při vkládání těchto deformovaných struktur (více než v testu 3.3) někdy algoritmus nekonverguje zpět.

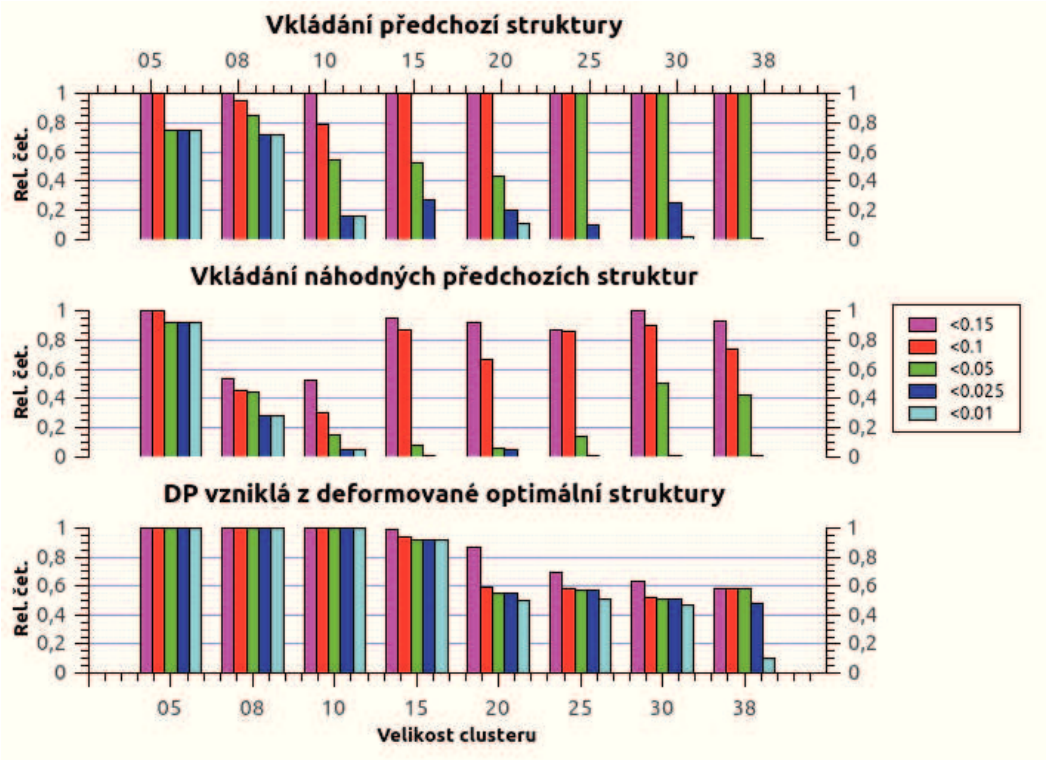


Obrázek 4: Srovnání výsledků optimalizace podle testu 3.1.2 pro počáteční populace (DP) s deformovanými předcházejícími strukturami a deformovanými náhodnými menšími strukturami doplněných náhodně umístěnými částicemi. Stejná míra deformace je použita i v posledním okně grafu, kdy je počáteční populace tvořena deformovaným známým globálním řešením.

Z výsledků plyne, že použitá deformace je natolik závažná, že při aplikaci na známé optimální struktury jsou výsledky optimalizace srovnatelné s optimalizací s náhodnou

počáteční populací (viz dále). Zajímavým výsledkem je ovšem zlepšení kvality optimalizace uplatněním stejné deformace při vkládání známých menších struktur. Toto může být dáno faktem, že pokud uplatníme deformaci na optimální strukturu, nutně snížíme její kvalitu, můžeme narušit výstavbové pravidlo dané struktury, ovšem při aplikaci na strukturu, která optimální není, může dojít i k náhodnému zlepšení kvality nebo náhodnému nalezení struktury podobné některému lokálnímu extrému.

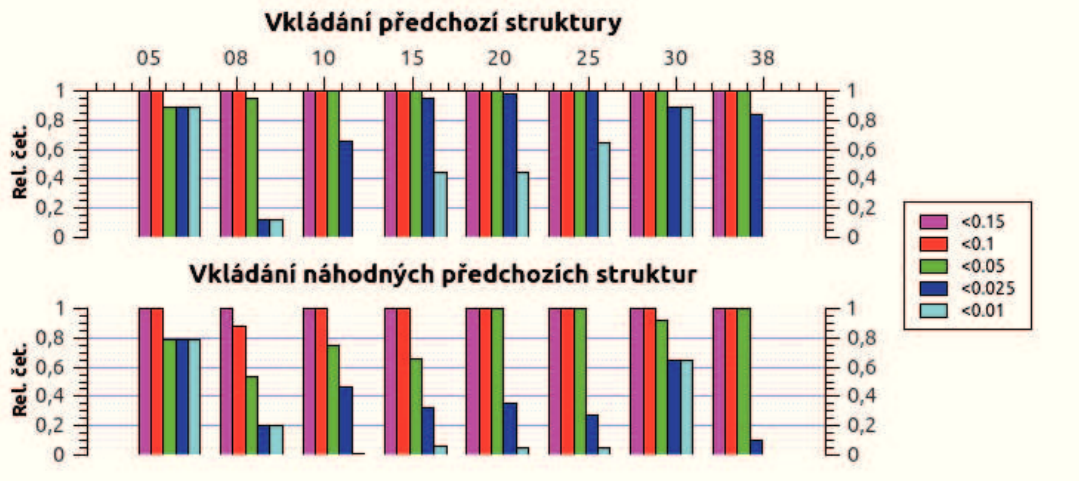
Vliv může mít také pravidlo pro ošetření bodů nacházející se mimo prohledávanou množinu. Pokud deformovaná optimální struktura opustí prohledávanou množinu, je uplatněn pružný odraz popsaný v kapitole 2.3.2. Daná částice může být umístěna do těsné blízkosti dalších částic, což způsobí navýšení potenciální energie a významné snížení kvality. Menší optimální struktury zaujímají menší objem ve středu prohledávaného prostoru s průměrem podle vzorce 15 a pravděpodobnost tohoto degenerativního odrazu je tedy nižší.



Obrázek 5: Srovnání výsledků optimalizace podle testu 3.1.2 pro počáteční populace (DP) s deformovanými předcházejícími strukturami a deformovanými náhodnými menšími strukturami doplněných náhodně umístěnými částicemi. Míra deformace je v tomto testu poloviční v porovnání s testy na obrázku 4. Stejná míra deformace je použita i v posledním okně grafu, kdy je počáteční populace tvořena deformovaným známým globálním řešením.

Z těchto testů lze vyčíst, že snížením míry deformace jsme zamezili výrazné degeneraci kvality deformovaných optimálních struktur, jelikož z nich tvořená počáteční populace je schopna konvergovat zpět ke globálnímu optimu. Při generování počáteční populace z menších optimálních struktur je ovšem nepravděpodobné, že by algoritmus našel globální

extrém. Může to být způsobeno tím, že existuje rozsáhlé okolí takto deformovaných struktur, v němž se nacházejí kvalitnější struktury, a při iteracích algoritmu dochází převážně k prohledávání tohoto okolí velkou částí agentů z populace. Fáze globálního prohledávání proto může být potlačena a algoritmy ustrnou v některém lokálním minimu.



Obrázek 6: Srovnání výsledků optimalizace podle testu 3.1.2 pro počáteční populace (DP) s vloženými předcházejícími strukturami a náhodnými menšími strukturami. Při vkládání struktur není použita deformace.

Pokud neuplatníme deformaci při vkládání menších optimálních struktur, omezíme rozsah jejich okolí v prohledávaném konfiguračním prostoru, ve kterém se vyskytují kvalitnější řešení. Každá struktura vytvořená z menších lokálních optim má zpravidla lepší kvalitu, než stejná struktura po deformaci. Při uvážení principu elitismu (definice 2.14), je menší pravděpodobnost, že algoritmus bude prohledávat okolí tohoto bodu (podle geometrické pravděpodobnosti), jelikož se v něm nacházejí méně kvalitní řešení. Kvalitnější agenti si tedy uchovávají svá řešení déle než při užití deformace a nedochází k předčasné konvergenci. Takový algoritmus bude mít lepší vlastnosti z hlediska globálního prohledávání prostoru a větší pravděpodobnost nalezení globálního optima (v jehož spádové množině se menší optimální struktury téměř jistě nevyskytují, tedy jej nenalezneme prohledáváním jejich blízkého okolí.)

Nejlepší výsledky algoritmus podává bez uplatnění deformace známých menších struktur (obrázek 6). Ze všech obrázku 4, 5 a 6 vyplývá, že nejlepších výsledků, bez ohledu na míru deformace, dosáhneme vkládáním předchozí optimální struktury.

3.6 Vliv velikosti prohledávané části konfiguračního prostoru

Prohledávaná množina je při optimalizaci omezena. Cílem tohoto testu je pro zvolené algoritmy zjistit, jestli změna její velikosti ovlivní výkon algoritmu.

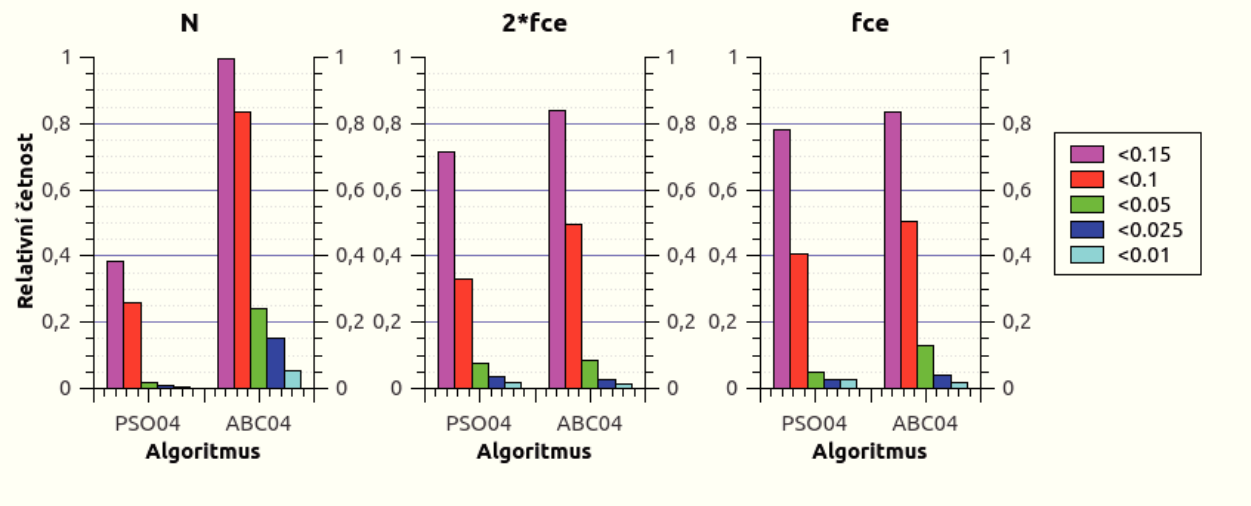
U počátečních verzí algoritmu byla prohledávaná množina nastavena příliš rozsáhlá, což vedlo k řešením obsahujícím disociované struktury. Následně proto byla vytvořena funkce prokládající průměry globálně optimalizovaných struktur převzatých z [1]. Základní verze nyní generuje struktury z hyperkrychle o délce hrany rovné dvojnásobku

funkce 15. Tyto jsou porovnávány s předchozí implementací pro hranu velikosti N a následnou úpravou, kdy se struktury generují z hyperkrychle o hraně délky proloženého průměru známých globálních řešení.

Průměry (největší vzdálenost mezi dvěma částicemi) byly počítány z převzatých optimálních struktur z [1]. Zhruba je prokládá funkce 15.

$$r = \sqrt{\frac{N}{10}} + \log N \tag{15}$$

kde N je počet částic systému.



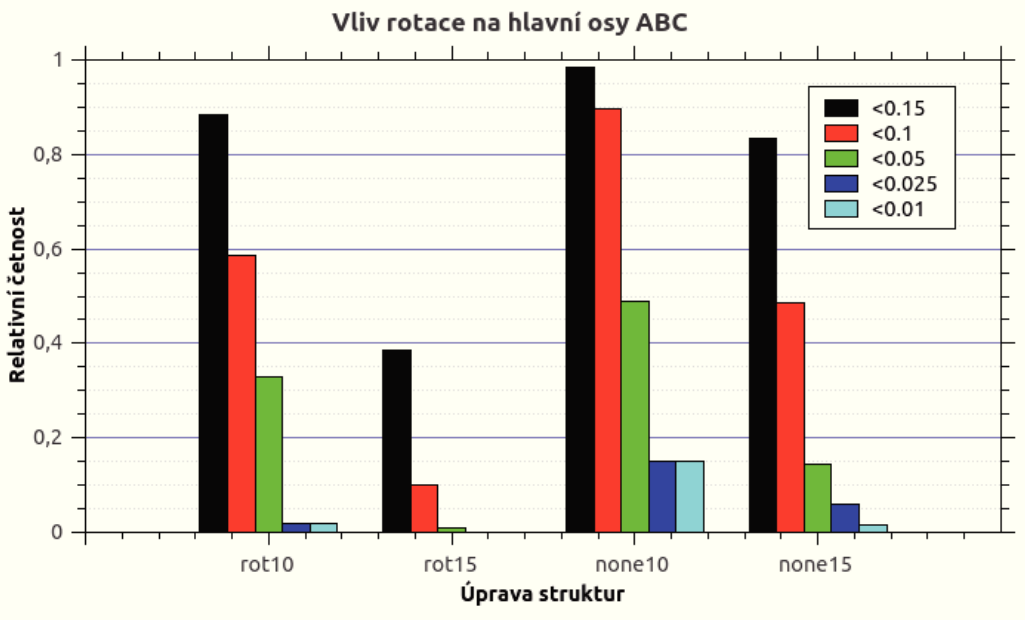
Obrázek 7: Výsledky optimalizace podle testu 3.1.2 pro algoritmy umělá včelí kolonie (ABC) a optimalizace hejnem částic (PSO) a klastr velikosti 15. Nadpis N značí výsledky pro hranu hyperkrychle délky N, u fce a $2 * fce$ je hrana krychle nastavena podle vzorce 15 resp. dvojnásobné hodnoty.

Z výsledků vyplývá, že ne pro všechny algoritmy je výhodné použít menší počáteční prostor. Algoritmus ABC má totiž z povahy své implementace omezené možnosti zvětšit průměr nalezené struktury (nová řešení se, až na průzkumníky, tvoří na spojnici dvou řešení z populace), tedy prohledává spíše „dovnitř“, zatímco algoritmus PSO využívá upravené techniky náhodného prohledávání a proto se může vymanit z počátečních omezení.

Při stanovení hranice prohledávané množiny je tedy třeba znát základní rysy vnitřní dynamiky daného algoritmu.

3.7 Vliv uplatnění rotace na osy setrvačnosti

Test srovnává výkon zvolených algoritmu před a po uplatnění principu rotace na hlavní osy setrvačnosti. Samotná aplikace této heuristiky si vyžádá nárůst výpočetního výkonu kvůli hledání vlastních vektorů tenzoru setrvačnosti. Test byl proveden pro klastry velikostí 10 a 15, algoritmus ABC s náhodným generováním počáteční populace a velikostí prohledávaného prostoru podle vzorce 15.



Obrázek 8: Výsledky optimalizace podle testu 3.1.2 pro algoritmus umělá včelí kolonie (ABC) a klastry velikosti 10 a 15. rotxx určuje výsledek pro klastr velikosti xx s využitím rotace na vlastní osy setrvačnosti, výsledky nonexx jsou získány bez uplatnění rotace na vlastní osy.

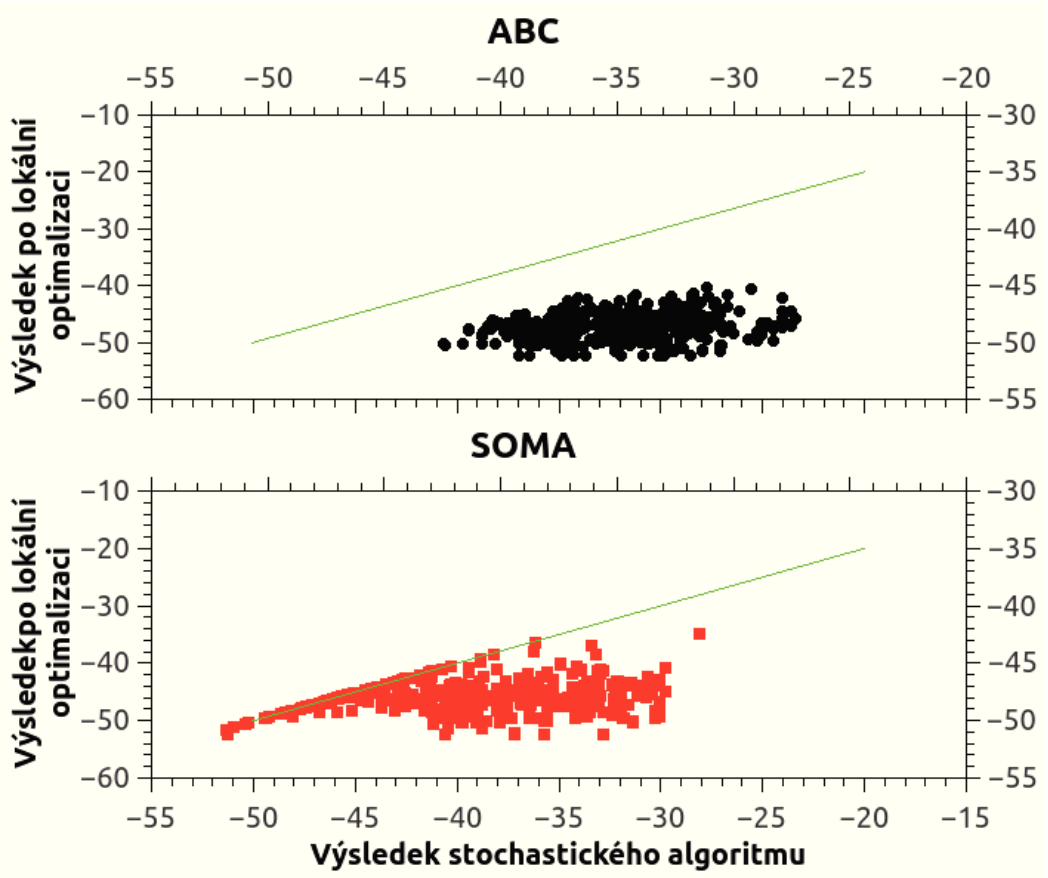
Algoritmus ABC při daném nastavení nezlepší svůj výkon po uplatnění této heuristiky. Toto může být opět dáno jeho povahou, kdy nová řešení se tvoří na spojnici dvou řešení z populace. Pokud struktury natočíme, aby si byly co nejpodobnější, můžeme znehodnotit u tohoto algoritmu fází tvoření nových řešení (při optimálním natočení povede spojnice počátkem soustavy souřadnic). Tato úprava se tedy jeví jako perspektivnější pro algoritmy, které nová řešení tvoří formou náhodného prohledávání (MCS). Je tedy na místě provést testy i pro jiné algoritmy.

3.8 Srovnání implementovaných algoritmů

Výsledky implementovaných algoritmů a jejich nastavených parametrů jsou shrnuty v následujících odstavcích. Nezabývám se určováním superiorního algoritmu.

3.8.1 Význam lokální optimalizace

Pro algoritmy ABC a SOMA je vykreslen výsledek testu 3.1.3. Z obrázku 9 vyplývá, že finální lokální optimalizace je esenciálním prvkem globální optimalizace. Ovšem některé algoritmy, speciálně algoritmus SOMA (jehož nová řešení se tvoří v okolí nejlepšího řešení v populaci), se často ukončí v lokálně optimalizované struktuře, proto jej používám jako čistě bez-gradientní metodu ve své implementaci hledání optimálního kroku při lokální optimalizaci (kapitola 2.1.1) nebo pro implementaci meta-optimalizace.

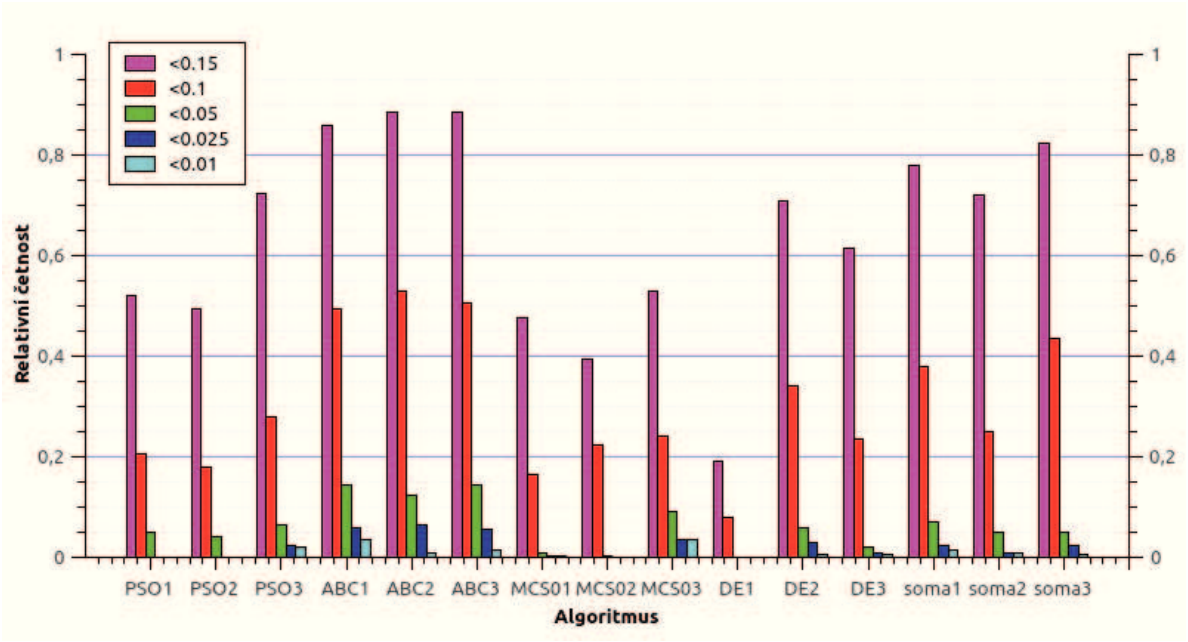


Obrázek 9: Grafické znázornění fáze dohledávání pro algoritmy umělá včelí kolonie (ABC) a samo-organizační migrační algoritmus (SOMA). Zelená křivka značí identickou funkci.

3.8.2 Srovnání algoritmů

Pro všechny implementované algoritmy a nastavení jejich parametrů jsou na obrázku 10 vykresleny výsledné statistiky podle testu 3.1.2. Testy probíhaly s náhodnou počáteční populací, s oblastí konfiguračního prostoru omezenou podle dvojnásobné hodnoty vzorce 15 a posunem do těžiště.

Slabší výkon algoritmu MCS může být způsoben použitím logistického rozdělení místo Lévyho rozdělení, které má diametrálně odlišné vlastnosti, ve fázi lokálního prohledávání.



Obrázek 10: Srovnání implementovaných algoritmů podle testu 3.1.2 s nastavením parametrů podle tabulek 1,2,3,4 a 5 pro klastr velikosti 15 s náhodnou počáteční populací.

Dále byl, vzhledem k získaným výsledkům, proveden test pro všechny implementované algoritmy a vybrané velikosti klastru. Nastavením počáteční generace probíhá vložením jedné předchozí menší struktury s přidáním náhodného atomu a zbytek populace je tvořen náhodně vybranými menšími strukturami s přidáním náhodně generovaných atomů pro zajištění diverzity (struktury převzaty z [1] a vkládány bez deformací). Prohledávaný prostor je omezen dvojnásobkem funkce 15 a je uplatněno posunutí do těžiště.

Tabulky 6 a 7 popisují počet algoritmů z 1000 nezávislých běhů, které nalezly výsledek v relativní vzdálenosti menší než 1% vzhledem ke známému řešení z [1] pro vybrané velikosti klastrů. Tabulky 8 a 9 udávají nejlepší nalezené řešení z 1000 nezávislých běhů daných algoritmů a porovnávají jej se známým řešením převzatým z [1].

4 Závěr

Cílem práce bylo implementovat balík fortranovských modulů, který by se dal využít pro řešení úlohy optimalizace atomových komplexů. Optimalizační úlohy byly řešeny pomocí stochastických algoritmů hejnové inteligence. Bylo navrženo několik rozšíření pro využití při řešení problému strukturální optimalizace a následně byl testován vliv těchto rozšíření na výsledky optimalizace vybraných algoritmů.

Schopnost algoritmů hejnové inteligence řešit vysoko dimenzionální optimalizační problémy molekulové fyziky byla prokázána v posledních letech na algoritmech PSO [11–13], ABC [42] a DE [38], stále ovšem zůstává prostor pro další rozvoj těchto metod a použitých vylepšení.

Řešenou úlohou v práci byla minimalizace potenciální energie struktury identických atomů s energií popsanou Lennardovým-Jonesovým potenciálem. Byly testovány algoritmy ABC [33], DE [37], PSO [10], MCS [41] a SOMA [6]. Výsledný programový balík byl vyvíjen s důrazem na modularitu pro následnou aplikaci na jiných úlohách a interakčních modelech nebo s využitím jiných algoritmů a jejich rozšíření. Základními testy pro porovnávání kvality algoritmů byla nejnižší nalezená hodnota ze stanoveného počtu nezávislých běhů a rozdělení vzdáleností funkčních hodnot řešení nalezenými jednotlivými běhy od známých funkčních hodnot převzatých z [1].

Výsledky testů uvedené v kapitole 3 potvrdily, že pro základní algoritmy a úpravy, které jsem použil, je náročné reprodukovat globální minima i pro systémy malé a střední velikosti [42]. Pro zajištění fungování programu i pro větší systémy je tedy třeba do základních algoritmů zahrnout specifická vylepšení zvyšující jejich účinnost při aplikaci na atomové a molekulové komplexy. Velikost prohledávané množiny a generování počáteční populace, zvláště vkládání menších nalezených struktur, se ukázaly jako významná nastavení algoritmu. Existuje ale mnoho dalších úprav, které se mohou projevit obdobně.

Některé modifikace jsou známy z publikované literatury [11–13, 17–26, 35, 38, 42]. Ze znalostí problému můžeme vyvodit řadu heuristických poznatků, které mohou vést k efektivnějšímu prohledávání konfiguračního prostoru. V molekulové fyzice je například možno využít některých poznatků o symetrii známých struktur pro generování nových řešení [11,38] nebo vytvoření metriky pro posuzování blízkosti dvou různých struktur pro následné rozdělení do paralelních hejn [11] (viz dále). Další možností je otestovat vhodnost různých souřadnicových soustav pro reprezentaci geometrické struktury studovaného systému (polární souřadnice, Z-matice, využití permutační symetrie potenciální energie systému pro identické částice).

Další možností je využití jiných algoritmů, jejich kombinace, případně syntéza nového algoritmu. V práci bylo posouzeno pouze několik vybraných z rozsáhlé množiny známých hejnových algoritmů. Literatura poukazuje na několik dalších základních algoritmů jako například: rojení světlušek [44], netopýří algoritmus [43], gravitační algoritmus [45], nabité částice [46], hejna kobylek [47, 48] nebo vlny hejnových částic [14–16], stejně tak jako na jejich kombinace a úpravy. Tyto opět využívají více agentů a definují jejich interakce v závislosti na kvalitě jimi nalezených řešení. Úkolem do budoucna tedy je porovnat úspěšnost širší množiny těchto algoritmů s optimálně nastavenými řídícími parametry a určit, který se nejlépe hodí pro řešení problémů z oblasti fyziky atomových a molekulových klastrů a které úpravy pro dané algoritmy jistě přispějí k urychlení a zkvalitnění řešení.

Teorie paralelních hejnových algoritmů [6, 11, 14–16, 38] je dalším krokem navazujícím na zavedení multiagentových algoritmů. Zatímco agentové systémy tvoří jednotnou populaci, kde členové interagují navzájem, paralelní hejna staví komunikační rozhraní mezi více populací a určuje pravidla pro interakci mezi nimi. Přímočarým rozšířením je jednoduché předávání nejlepšího řešení mezi populacemi. Z jedné populace tak vymizí řešení, které ji mohlo držet v lokálním minimu, zatímco když jej vložíme do populace jiné, může vést k prohledávání jiné části konfiguračního prostoru. Existuje celá řada pokročilejších pravidel pro interakci mezi populacemi a podobně jako v hejnových algoritmech se i zde často hledá inspirace v živé přírodě a organizovaných společenstvích.

Úkolem jednodušších interakčních modelů je aproximace výsledků zjednodušením interakčního modelu pro modely složitější a přesnější. Výsledky z těchto modelů mohou dopomoct při nastavení parametrů při meta-optimalizaci nebo poukázat na kvalitu užitých vylepšení. V budoucnu je plánováno využití algoritmů pro složitější interakční modely právě po uplatnění informací získaných z modelů jednodušších.

Podle literatury [11, 14–16, 29–32, 38, 38], která využívá průběžnou lokální optimalizaci řešení nalezenými jednotlivými agenty, vyžadují základní algoritmy řádově tisíce i více lokálních optimalizací v průběhu jednoho běhu algoritmu. I v provedených testech bez průběžné lokální optimalizace se také ukázal strmý nárůst nutného počtu ohodnocení účelové funkce s rostoucí velikostí systému. Intuitivně se tedy jako nutná součást těchto algoritmů jeví jejich paralelizace. V současné verzi je využito direktiv OpenMP. Zdá se být vhodné v budoucnu využít hybridní paralelizace (OpenMP, MPI (message passing interface) a GPGPU (general-purpose graphics processing unit)).

Před testováním algoritmů při aplikaci na jiné modely a úlohy je třeba vyřešit některé problémy, které vyvstaly při základním průzkumu oblasti stochastických optimalizačních algoritmů. Jedním problémem pro další posuzování kvality algoritmů a jejich modifikací je zpracování dat. Mnoho různých algoritmů a jejich rozšíření znamená i mnoho dat k analýze. V práci, je posuzováno jen několik vybraných kombinací algoritmů a užitých vylepšení pomocí testů popsaných v kapitole 3. Ideální je ovšem prověřit všechny algoritmy vůči všem možným úpravám. Jako první krok se tedy jeví návrh datové struktury pro přehledné a nejlépe automatické zpracování velkého množství dat, jelikož v práci bylo zmíněno pouze několik z, s nadsázkou řečeno, nekonečně mnoha možných vylepšení.

Veškerá srovnání algoritmů v mé práci probíhala pouze na základě statistik vytvořených z nalezených výsledků, ovšem na samotný algoritmus je možné pohlížet jako na dynamický proces. V takovémto případě se nabízí otázka, jaké parametry můžeme v průběhu iterací algoritmu měřit. U většiny numerických metod je základním ukazatelem rychlost konvergence. I u stochastických algoritmů můžeme vytvořit statistiku průběžných nalezených řešení a rychlost konvergence určovat z jejího vývoje. Není to ovšem jediný indikátor. Může nás také zajímat co se děje s jednotlivými agenty. Pak narážíme na paralely s termodynamikou a můžeme zkusit zavést střední kvadratické rychlosti a následně vnitřní energii agentového systému. Případně nás může zajímat, jak daný algoritmus prohledává uvažovaný prostor, tedy jak jsou po něm agenti rozmístěni a jak toto rozmístění mění během jednotlivých iterací. Dalším krokem v práci se stochastickými algoritmy by tedy mohl být pokus jejich porovnání vzhledem k vnitřnímu chování. Zjistit, které vlastnosti můžeme kvantifikovat a na kterých závisí kvalita výsledného řešení.

5 Reference

[1] Cambridge cluster database, *http://www-wales.ch.cam.ac.uk/CCD.html*

[2] *D. J. Wales, Energy Landscapes. Cambridge University Press, 2003, 654 s. ISBN 0-521-81415-4*

[3] *Z. Dostál, P. Beremlijski, Metody Optimalizace, 2012*

[4] *J. Bouchala, O. Vlach, Křivkový a plošný integrál, 2012*

[5] *J. Kuben, Š. Mayerová, P. Račková, P. Šarmanová, Diferenciální počet funkcí více proměnných, 2012*

[6] *I. Zelinka, Z. Oplatková, M. Šeda, P. Ošmera, F. Včelař, Evolutionary techniques – principles and applications, BEN, Prague, 2008, 598 s. ISBN 80-7300-218-3*

[7] *S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Science, 220:671–680, 1983*

[8] *I. Rechenberg, Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution (PhD thesis). Reprinted by Fromman-Holzboog (1973)*

[9] *J. Kvasnica, A. Havránek, P. Lukáč, B. Sprušil, Mechanika. Vyd. 2. Praha: Academia, 2004, 476 s. ISBN 80-200-1268-0*

[10] *J. Kennedy,R. Eberhart, Particle swarm optimization, IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS PROCEEDINGS, VOL 5 (1995) 1942-1948*

[11] *J. LV, Y.C. Wang, L. Zhu, Y.M. MA, Particle-swarm structure prediction on clusters, Journal of Chemical Physics, 137 (2012) 084104*

[12] *S.T. Call, D.Y. Zubarev, A.I. Boldyrev, Global minimum structure searches via particle swarm optimization, Journal of computational chemistry, 28 (2007) 1177-1186*

[13] *Y.C. Wang, J. LV, L. Zhu, Y.M. MA, CALYPSO: A method for crystal structure prediction, Computer physics communication, 183 (2012) 2063-2070*

[14] *T. Hendtlass, WoSP: A multi-optima particle swarm algorithm, iEEE congress on evolutionary computation, Vols 1-3 (2005) 727-734*

[15] *T. Hendtlass, A Particle Swarm algorithm for high dimensional, multi-optima problem, ieee swarm intelligence symposium, (2005) 149-154*

[16] *T. Hendtlass, A particle swarm algorithm for complex quantised problem spaces, IEEE congress on evolutionary computation, Vols 1-6 (2006) 1000-1004*

[17] *L.W. Sai, J.J. Zhao, X. Huang, J. Wang, Structural evolution and electronic properties of medium-sized gallium clusters from ab initio genetic algorithm search, Journal of Nanoscience and Nanotechnology, 12 (2012) 132-137*

[18] *J.J. Zhao, R.H. Xie, Genetic Algorithms for the geometry optimization of atomic and molecular clusters, Journal of Computational and Theoretical Nanoscience, 1 (2004) 117-131*

[19] *N. Dugan, S. Erkoc, Genetic algorithm-Monte Carlo hybrid geometry optimization method for atomic clusters, Science and Engineering, 45 (2009) 127-132*

[20] *B. Hartke, Application of evolutionary algorithms to global cluster geometry, Applications of evolurionary computation in chemistry, 110 (2004) 33-53*

[21] *J.O. Joswig, M. Springborg, Genetic-algorithms search for global minima of aluminum clusters using a Sutton-Chen potential, Physical review B, 68 (2003) 085408*

[22] *R.L. Johnston, Evolving better nanoparticles: Genetic algorithms for optimising cluster, Dalton transactions, IS:22 (2003) 4193-4207*

[23] *P. Bobadova-Parvanova, K.A. Jackson, Scanning the potential energy surface of iron clusters: A novel search strategy, Journal of chemical physics, 116 (2002) 3576-3567*

[24] *C. Roberts, R.L. Johnston, A genetic algorithm for the structural optimization of Morse clusters, Theoretical chemistry accounts, 104 (2000) 123-130*

[25] *C. Barron, S. Gomez, A genetic algorithm for Lennard-Jones atomic clusters, Applied mathematics letters, 12 (1999) 85-90*

[26] *W.J. Pullan, Genetic operators for the atomic cluster problem, Computer physics communication, 107 (1997) 137-148*

[27] *W.S. Cai, Y. Feng, Optimization of Lennard-Jones atomic clusters, Journal of molecular structure – theochem, 579 (2002) 229-234*

[28] *R. Fournier, S. Bulusu, S. Chen, J. Tung, Using swarm intelligence for finding transition states and reaction paths, Journal of Chemical Physics, 135 (2011) 104117*

[29] *S.E. Schonborn, S. Goedecker, The performance of minima hopping and evolutionary algorithms for cluster structure prediction, Journal of chemical physics, 130 (2009) 144108*

[30] *R.H. Leary, Global optimization on funneling landscapes, Journal of global optimization, 18 (2000) 367-383*

[31] *J.P.K. Doye, D.J Wales, Thermodynamics and the global optimization of Lennard-Jones clusters, Journal of chemical physics, 109 (1998) 8143-8153*

[32] *D.J. Wales, J.P.K. Doye, Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms, Journal of physical chemistry a, 101 (1997) 5111-5116*

[33] *D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, Journal of Global Optimization, 39 (2007) 459-471*

[34] *C. Schiffmann, D. Sebastiani, Artificial Bee Colony Optimization of Capping Potentials for hybrid Quantum Mechanical/molecular mechanical calculations, Journal of Chemical Theory and Computation, 7 (2011) 1307-1315*

[35] *M. Sonmez, Artificial Bee Colony algorithm for optimization of truss structures, Applied soft computing, 11 (2011) 2406-2418*

[36] *D. Karaboga, B. Akay, A survey: algorithms simulating bee swarm intelligence, Artificial intelligence review, 31 (2009) 61-85*

[37] *R. Storn, K. Price, Differential Evolution-A simple and efficient adaptive scheme for global optimization over continuous spaces, 1995*

[38] *C. Zhangui, J. Xiangwei, L. Jingbo, L. Shushen, W. Linwang, PDECO: Parallel differential evolution for clusters optimization, Journalof Computational Chemistry, 34 (2013) 1046-1059*

[39] *E. Zitzler, K. Deb, L. Thiele, Comparison of multiobjective evolutionary algorithms: empirical results, Institut fur technische informatik und kommunikationsnetze, ETH Zurich, TIK-report 70*

[40] *X.S. Yang, S. Deb, Cuckoo search via Lévy Flights, World congress on nature & biologically inspired computing, 2009 210-214*

[41] *S. Walton, Modified cuckoo search: A new gradient free optimisation algorithm, Chaos, Solitons & Fractals, 44 (2011) 710-718*

[42] *C. Wehmeyer, G.F. von Rudorff, D. Sebastiani, Foraging on the potential energy surface: A swarm intelligence-based optimizer for molecular geometry, Journal of Chemical Physics, 137 (2012) 194110*

[43] *X.-S. Yang, A New Metaheuristic Bat-Inspired Algorithm, in: Nature Inspired Cooperative Strategies for Optimization (NICSO 2010) (Eds. J. R. Gonzalez et al.), SCI 284, 65-74 (2010)*

[44] *L.S. Coelho, D.L.A. Bernet, V.C. Mariani, A chaotic firefly algorithm applied to reliability-redundancy optimization, 2011 IEEE Congress on Evolutionary Computation (CEC), 517-521*

[45] *E. Rashedi,H. Nezamabadi-pour, S. Saryazdi,GSA: A gravitational search algorithm, Information Sciences 179 (2009) 2232-2248*

[46] *A. Kaveh, S. Talatahari, A novel heuristic optimization method: charged system search, Acta Mech 213, 267-289 (2010)*

[47] *S. Chen, Locust swarms - a new multi-optima search technique, 2009 IEEE Congress on Evolutionary Computation (CEC 2009), 1745-1752*

[48] *S. Chen, An Analysis of Locust Swarms on Large Scale Global Optimization Problems ,In: K. Korb, M. Randall, and T. Hendtlass (Eds.): ACAL 2009, LNAI 5865, pp. 211–220, 2009*

A Tabulky

velikost	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
ABC01	763	8	8	345	2	77	3	4	0	45	0	0	37	0	0
ABC02	612	5	0	386	11	78	2	2	1	18	0	12	46	0	0
ABC03	668	2	0	445	9	92	3	1	1	102	8	18	35	0	0
DE01	661	9	0	642	18	250	30	37	12	327	10	37	148	0	18
DE02	1,000	0	0	697	91	148	0	23	17	175	1	7	55	0	14
DE03	938	1	0	329	5	85	1	48	14	137	39	29	145	0	0
MCS01	731	0	0	668	2	97	7	83	0	308	8	0	24	0	0
MCS02	680	0	0	669	0	20	0	5	35	159	0	1	0	0	0
MCS03	543	8	0	580	0	149	0	0	0	229	2	0	160	0	0
SOMA01	879	0	0	834	0	243	16	17	31	512	35	35	178	0	0
SOMA02	984	35	0	766	41	132	9	13	0	152	18	1	51	0	7
SOMA03	959	24	0	943	32	274	32	16	0	814	8	32	256	0	0
PSO01	962	24	0	799	19	168	11	19	8	225	27	26	128	0	5
PSO02	904	19	0	679	0	167	7	13	7	211	18	30	93	0	0
PSO03	790	2	0	627	1	130	5	1	3	191	3	5	111	0	1

Tabulka 6: Tabulka výsledků popisuje počet z 1000 nezávislých běhů pro algoritmy popsané v kapitole 3.2 a zvolené velikosti systému, jež našly řešení x splňující $\frac{|f(x)-CCD_{min}|}{|CCD_{min}|} < 0.01$, kde CCD_{min} je hodnota převzata z [1], část 1.

velikost	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
ABC01	8	0	0	0	0	9	0	8	11	0	17	0	0	0	0
ABC02	1	13	0	0	28	0	0	18	0	0	2	8	0	0	3
ABC03	0	3	0	0	54	22	0	29	0	0	17	16	0	9	8
DE01	36	98	13	0	97	20	3	51	32	6	101	0	1	11	0
DE02	38	16	5	1	27	0	0	22	0	0	44	0	0	2	23
DE03	30	6	4	0	19	10	0	7	23	0	16	0	0	3	0
MCS01	2	15	1	0	6	66	0	0	40	0	173	0	0	0	0
MCS02	22	4	21	0	328	0	0	7	8	0	16	0	0	0	0
MCS03	240	96	0	0	0	149	0	0	0	0	3	0	0	0	0
SOMA01	62	86	39	0	122	125	3	174	64	1	100	8	0	10	35
SOMA02	53	59	3	0	53	26	0	10	29	0	25	10	0	8	12
SOMA03	180	206	74	0	166	123	0	144	33	0	78	16	0	19	10
PSO01	47	41	16	2	49	29	0	70	23	0	55	0	0	4	17
PSO02	43	22	13	5	63	23	14	66	36	0	122	20	10	24	4
PSO03	18	33	5	0	41	18	0	63	12	0	77	0	0	9	2

velikost	35	36	37	38	39	40
ABC01	0	0	5	0	0	8
ABC02	0	0	0	0	0	1
ABC03	0	0	4	0	0	0
DE01	1	22	79	0	0	29
DE02	6	5	1	0	0	24
DE03	3	13	1	0	0	19
MCS01	0	0	0	0	0	14
MCS02	0	0	1	0	0	0
MCS03	0	0	0	0	0	0
SOMA01	0	0	36	0	0	67
SOMA02	0	7	0	0	0	13
SOMA03	32	0	38	0	0	33
PSO01	5	4	35	0	0	37
PSO02	8	0	44	0	5	32
PSO03	0	0	15	0	6	18

Tabulka 7: Tabulka výsledků popisuje počet z 1000 nezávislých běhů pro algoritmy popsané v kapitole 3.2 a zvolené velikosti systému, jež našly řešení x splňující $\frac{|f(x)-CCD_{min}|}{|CCD_{min}|} < 0.01$, kde CCD_{min} je hodnota převzata z [1], část 2.

velikost	5	6	7	8	9	10	11	12	13	14	15	16	17	18
CCDm	−9.1	−12.71	−16.51	−19.82	−24.11	−28.42	−32.77	−37.97	−44.33	−47.85	−52.32	−56.82	−61.32	−66.53
ABC01	−9.1	−12.71	−16.5	−19.82	−24.11	−28.42	−32.74	−37.96	−41.45	−47.82	−51.39	−55.86	−61.29	−65.72
ABC02	−9.1	−12.71	−15.93	−19.82	−24.11	−28.42	−32.72	−37.91	−44.24	−47.81	−51.38	−56.78	−61.29	−65.8
ABC03	−9.1	−12.71	−15.93	−19.82	−24.11	−28.42	−32.73	−37.89	−44.29	−47.84	−52.29	−56.78	−61.28	−65.75
DE01	−9.1	−12.71	−15.93	−19.82	−24.11	−28.42	−32.75	−37.95	−44.31	−47.84	−52.29	−56.78	−61.29	−62.8
DE02	−9.1	−12.3	−15.93	−19.82	−24.11	−28.42	−28.82	−37.94	−44.31	−47.83	−52.27	−56.78	−61.29	−65.64
DE03	−9.1	−12.71	−15.93	−19.82	−24.11	−28.42	−32.71	−37.95	−44.31	−47.83	−52.31	−56.79	−61.29	−65.77
MCS01	−9.1	−12.3	−15.94	−19.82	−24.1	−28.42	−32.75	−37.96	−41.46	−47.83	−52.27	−55.89	−61.29	−65.71
MCS02	−9.1	−12.3	−15.93	−19.82	−23.17	−28.42	−31.88	−37.95	−44.31	−47.83	−51.35	−56.75	−60.36	−65.74
MCS03	−9.1	−12.71	−15.93	−19.82	−23.15	−28.42	−31.88	−36.13	−41.45	−47.83	−52.29	−55.82	−61.29	−65.16
SOMA01	−9.1	−12.3	−15.93	−19.82	−23.23	−28.42	−32.75	−37.95	−44.31	−47.83	−52.31	−56.8	−61.29	−65.8
SOMA02	−9.1	−12.71	−15.93	−19.82	−24.11	−28.42	−32.74	−37.94	−41.47	−47.82	−52.29	−56.75	−61.29	−65.69
SOMA03	−9.1	−12.71	−15.94	−19.82	−24.11	−28.42	−32.76	−37.94	−41.47	−47.85	−52.28	−56.81	−61.3	−65.84
PSO01	−9.1	−12.71	−15.94	−19.82	−24.11	−28.42	−32.74	−37.96	−44.31	−47.83	−52.3	−56.79	−61.28	−65.77
PSO02	−9.1	−12.71	−15.93	−19.82	−23.23	−28.42	−32.74	−37.95	−44.29	−47.83	−52.3	−56.79	−61.28	−65.75
PSO03	−9.1	−12.71	−15.93	−19.82	−24.1	−28.42	−32.73	−37.89	−44.3	−47.83	−52.3	−56.77	−61.29	−65.72

velikost	19	20	21	22	23	24	25	26	27	28	29	30	31
CCDm	−72.66	−77.18	−81.68	−86.81	−92.84	−97.35	−102.37	−108.32	−112.87	−117.82	−123.59	−128.29	−133.59
ABC01	−69.97	−77.01	−80.28	−85.93	−91.09	−96.26	−101.59	−105.98	−112.59	−117.58	−121.78	−127.85	−131.22
ABC02	−70.98	−76.57	−81.4	−85.84	−91.16	−97.1	−97.48	−106.52	−112.66	−116.12	−121.31	−127.86	−132.77
ABC03	−69.97	−76.28	−81.36	−85.83	−91.09	−97.19	−102.05	−106.82	−112.53	−116.44	−122.1	−127.92	−132.92
DE01	−72.49	−77.06	−81.54	−86.08	−91.27	−97.23	−101.73	−108	−112.65	−117.45	−123.13	−128	−130.13
DE02	−72.59	−77.09	−81.44	−86.38	−92.24	−97.21	−101.3	−106.57	−112.61	−115.94	−122.02	−127.93	−131.87
DE03	−70.88	−76.95	−81.51	−86.04	−91.23	−96.93	−102.14	−106.9	−112.15	−117.25	−122.2	−127.89	−129.09
MCS01	−67.16	−77.02	−81.53	−86.03	−89.8	−97.04	−101.94	−106.86	−111.32	−117.21	−121.74	−127.8	−129.99
MCS02	−70.92	−77.01	−81.26	−86.08	−90.77	−97.21	−97.64	−102.42	−112.33	−116.93	−122.22	−127.92	−131.56
MCS03	−70.11	−77.1	−81.62	−85.53	−86.82	−95.75	−101.89	−105.3	−111.47	−115.86	−122.07	−127.65	−130.53
SOMA01	−71.03	−77.08	−81.57	−86.5	−91.19	−97.27	−101.79	−107.78	−112.72	−117.29	−122.98	−127.96	−132.52
SOMA02	−72.54	−77.08	−81.57	−86.06	−91.17	−97.14	−102.09	−106.83	−112.73	−117.6	−122.01	−127.91	−132.91
SOMA03	−71.08	−77.17	−81.67	−86.2	−91.24	−97.3	−101.95	−106.73	−112.81	−117.32	−122.19	−128.12	−133.13
PSO01	−72.48	−77.07	−81.49	−86.24	−92.41	−97.15	−102	−106.6	−112.72	−117.15	−122.17	−127.97	−131.95
PSO02	−70.97	−77.02	−81.5	−86.73	−92.45	−97.2	−102.01	−107.9	−112.59	−117.39	−122.1	−128.01	−132.84
PSO03	−72.25	−77.05	−81.55	−86.04	−91.14	−97.17	−102.13	−106.62	−112.68	−117.07	−122.06	−127.93	−132.19

Tabulka 8: Tabulka výsledků popisuje nejlepší nalezená řešení z 1000 nezávislých běhů algoritmu s nastavením parametrů podle tabulek z kapitoly 3.2 pro zvolené velikosti systému a srovnává jej se známými funkčními hodnotami v globálních řešeních převzatých z [1], část 1.

velikost	32	33	34	35	36	37	38	39	40
CCDm	−139.64	−144.84	−150.04	−155.76	−161.83	−167.03	−173.93	−180.03	−185.25
ABC01	−137.2	−143.1	−147.01	−152.86	−159.41	−166.29	−171.79	−177.77	−183.58
ABC02	−137.08	−143.17	−148.83	−153.61	−158.86	−164.5	−171.81	−177.31	−184.3
ABC03	−137.3	−144.28	−148.77	−153.93	−159.4	−165.47	−171.17	−177.68	−182.75
DE01	−139.24	−144.32	−147.42	−154.31	−161.68	−166.8	−172.15	−177.96	−183.7
DE02	−136.29	−143.48	−148.85	−155.12	−161.7	−165.58	−170.14	−178.1	−183.97
DE03	−137.6	−144.55	−148.03	−154.86	−160.88	−165.5	−169.98	−178.04	−183.71
MCS01	−134.94	−141.87	−146.62	−153.99	−159.21	−162.33	−169.01	−177.76	−183.9
MCS02	−133.95	−139.66	−148.03	−150.33	−155.78	−166.3	−170.28	−174.42	−180.24
MCS03	−134.24	−139.74	−147.45	−153.53	−156.66	−165.24	−171.2	−177.42	−182.59
SOMA01	−137.55	−143.59	−149.59	−153.81	−159.59	−166.92	−172.19	−177.99	−184.09
SOMA02	−137.22	−143.8	−149.08	−153.68	−160.27	−164.91	−172.05	−178.04	−183.99
SOMA03	−137.6	−143.77	−148.79	−154.79	−159.78	−166.9	−172.07	−178.01	−184.1
PSO01	−137.91	−144.57	−148.89	−154.89	−160.78	−166.82	−172.13	−177.92	−183.97
PSO02	−138.68	−144.57	−148.74	−154.6	−159.64	−166.85	−171.61	−178.49	−184.78
PSO03	−136.88	−144.01	−148.7	−154	−159.71	−166.8	−172.15	−178.53	−184.03

Tabulka 9: Tabulka výsledků popisuje nejlepší nalezená řešení z 1000 nezávislých běhů algoritmu s nastavením parametrů podle tabulek z kapitoly 3.2 pro zvolené velikosti systému a srovnává jej se známými funkčními hodnotami v globálních řešeních převzatých z [1], část 2.

B Programová implementace

Viz přiložené CD.